Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

PhD Thesis

Trinity College Dublin

The University of Dublin

School of Computer Science and Statistics

Recommender Systems: A Study of User-Cold Start and Reinforcement Learning

---

Author:
Dilina Chandika Rajapakse

Supervisor:
Professor Douglas Leith

2026

A thesis submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

I, the undersigned, agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signed: _____ Date: 15.12.2025

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Douglas Leith. I am immensely thankful for your continuous support and patience throughout my PhD journey. You have taught me how to conduct meaningful research and consistently encouraged me to explore new ideas and opportunities. I have always admired your unique perspective on problems and your insightful feedback. The knowledge and experience I gained during this period have been invaluable to my growth as a researcher. I am truly grateful to have had you as my supervisor.

A big Thank you to Vaibhav, David, Mohamed and Dr. Sulthana, who are not just peers in the lab, but also some of my closest friends throughout the years. I am grateful for the time we spent together and all the memories we made, both inside and outside the lab. I will surely miss all the banter and our table tennis sessions in the lab. I am glad that I was able to share this journey with you, and also be a part of your PhD journeys.

During my PhD, I met some wonderful people and fellow researchers. Special thanks to Ziyi, Tiangfang and Dr. Dipto. From talking about our research to dreaming of being food bloggers, I enjoyed the time we spent exploring different cuisines around Ireland. I also like to thank Daniel, Thulitha, Loshan and all my friends in Ireland, who made it feel a bit closer to home throughout these years. To my friends and family around the world, I am grateful for always keeping in touch and supporting me.

I am deeply thankful to Prof. Jannach Dietmar at University of Klagenfurt in Austria. Since the beginning of my PhD, I have admired your contributions to our research field, and I am glad to have had the opportunity to work closely with you during our research collaboration. To the colleagues at University of Klagenfurt, I am grateful for your warm welcome and support. I would also like to thank all the amazing people I met in Austria for making my stay there an enjoyable one.

I gratefully acknowledge the generous financial support provided by Research Ireland Centre for Future Networks (CONNECT) under Grant No. 16/IA/4610, which made this research possible. I am thankful for Trinity College Dublin and CONNECT Centre for their support during this journey.

Finally, my heartfelt thanks to my parents, brothers and grandparents for their unwavering support, encouragement and belief in me, even while being continents apart. None of this would have been possible without my loving family.

I express my gratitude to everyone who, directly or indirectly, contributed to the success of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A recommender system (RS) needs to accurately identify the preferences or interests of a user in order to provide personalized recommendations. This is particularly challenging with new users, where little is initially known about a user's preferences and so the recommender system must quickly learn these as the user interacts with it - a scenario known as the user-cold start problem in RS. Beyond cold-start, a user's behaviour on a system can change over time, typically reflecting the evolving preferences and interests of the user and a recommender needs to dynamically adapt to these changing behaviours. In this thesis, we identify the fundamental need to **continuously** learn an effective representation of a user's preferences. With that in mind, we explore the use of Reinforcement Learning (RL) techniques in addressing various item recommendation tasks and study the effectiveness of RL in recommenders.

Research on recommender systems and personalization can be dated back to the 1990s (Basu et al., 1998; Resnick & Varian, 1997) and applications of RS can be seen in many online services such as e-commerce, streaming platforms, social media, news, etc. A key milestone in RS was in 2006, where the Netflix Prize Challenge (Bennett & Lanning, 2007), an open competition organized by Netflix[1], garnered a lot of attention towards recommender systems. In this challenge, a large dataset of known user-item ratings is provided, where the goal of the recommender is to predict the unknown user ratings. Data with which recommender systems are trained, similar to this dataset, are highly sparse due to the large number of unknown or missing ratings, as most users interact with only a small fraction of the available items. Collaborative filtering (CF) became a popular approach in recommender systems (Su & Khoshgoftaar, 2009), for handling sparse datasets by leveraging similarities in user-item interactions. Matrix factorization (MF) was shown to be the most successful collaborative filtering approach in early works on recommender systems

[1] www.netflix.com

and was also the winning approach of the Netflix prize challenge.

Since then there have been significant changes in recommendation models - from traditional Matrix-factorization models to Deep-Learning models and Graph-based methods (Y. Li et al., 2024). At the same time, there has been an increase in recommender datasets with various types of information including different forms of feedback (i.e: ratings, clicks, reviews, etc) and additional information about users and items (Marcuzzo et al., 2022). Despite the growth and advancements in the field, the problem of providing better recommendations to a user still remains challenging.

Recommending items to new users has been a fundamental problem in recommender systems and numerous studies have addressed this user-cold start literature (H. Yuan & Hernandez, 2023). A common observation seen is that a new user (i.e: *cold-user*) is considered to have a minimum number of prior interactions to determine an initial user representation (e.g: (C. C. Chen et al., 2023; Feng et al., 2021)). These methods do not provide recommendations to new users with no history or a lesser number of prior interactions (i.e: pure cold start conditions). To mitigate this absence of user-item interaction data, a recommender can use additional side information[2] about these new users to recommend the first few items (Abdullah et al., 2021). However, this side information may not always be available to a system, for example, due to data protection acts or users not opting to provide information (Z. Chen et al., 2024). An alternative approach is to use non-personalized methods such as recommending popular or random items to new users (Rashid et al., 2002), but these do not offer valuable insights into a user's preferences.

It can be seen that many of the cold-start literature often rely on static representations of user preferences and do not adequately adapt to changing user behaviours. At the same time, there is an artificial disjoint between cold-start recommendations and *normal* operation of a recommender system. This means that providing recommendations for new users and existing warm users are handled as two separate modes, and there is a lack of research showing how these cold-start approaches adapt to provide recommendations in the long run, ideally in the same framework. Therefore in this thesis, we focus on continuously learning about user preferences and providing recommendations, starting from pure cold start conditions. We also do not use any side information about users or items.

We focus on a recommender setup where the system presents a user with a single-item, or a list of items and the user will then select an item out of the recommendations and provide feedback. We mainly consider this feedback to be an explicit feedback of an item-rating, but it could also be engagement time, like-dislike, etc. Using this information, the system then recommends the next item(s).

[2]This additional information includes contextual information about the users such as the gender, location, etc.

Therefore to make meaningful recommendations, learning an accurate representation of the user's preferences from interaction data is a crucial aspect. Additionally, a recommender should not only present items according to the current estimated preferences of the user, but also discover unknown interests of the user. In other words, the RS should *explore* the unknown preferences of the user, while providing meaningful recommendations by *exploiting* the known preferences.

In this thesis, we propose two recommendations approaches that carry out online learning based on Reinforcement Learning techniques. First, we introduce a novel Monte Carlo Tree Search (MCTS) based recommender, which is shown to quickly learn a new user's preferences. We then propose an RL-enhanced transformer-based recommender model *RLT4Rec*. We show that this recommender learns an informative user-state, and in an RL perspective the user representation (also referred to as the 'state') plays a major role in providing good recommendations. The proposed methods are primarily evaluated under user-cold start conditions, demonstrating the capability to learn a user's preferences with fewer interactions. Both the MCTS and RLT4Rec can cater to new and established users in the same framework as they can continuously learn about the users to provide recommendations. The final chapter of the thesis is a study that investigates the reproducibility and effectiveness of reinforcement learning (RL) based recommender systems in the existing sequential recommendation literature, specifically in myopic *next-item* prediction tasks. We find that there is a severe lack of reproducibility and also inconsistencies in the evaluations of RL-based recommender performance. We analyse the relative performance of published RL approches against simpler supervised learning based recommenders. This final study also highlights the necessity for the research community to improve the quality and thoroughness of the methods used in research and, in doing so, ensure consistent and reliable progress within the field.

1.1 Contributions of the Thesis

This thesis consists of three main research chapters. In this section, we provide a brief overview of these chapters and their contributions.

Chapter 2 - User Cold-Start Learning in Recommender Systems using Monte Carlo Tree Search: We first address the User-cold start problem in Recommender Systems using Monte Carlo Tree Search (MCTS), a model-based reinforcement learning technique. We show that MCTS quickly learn a user's preferences and achieves state of the art. MCTS is known to be inefficient with larger search spaces, etc, we attempt to address these shortcomings in order to reduce the computational time and make fast recommendations, without compromising the

accuracy.

To summarize the contributions in this Chapter:

1. Propose a novel approach for user-cold start learning using Monte-Carlo Tree Search recommender, based on clustered-user groups

2. We show how the MCTS can be adapted to different feedback forms beyond explicit item-ratings.

3. Provide an implementation of the proposed MCTS in a fast and efficient C++ framework

This chapter is based on the following publications:

- Dilina Chandika Rajapakse and Douglas Leith. 2022. Fast and Accurate User Cold-Start Learning Using Monte Carlo Tree Search. In Proceedings of the 16th ACM Conference on Recommender Systems (RecSys '22).

- Dilina Chandika Rajapakse and Douglas Leith. 2023. User Cold-Start Learning In Recommender Systems Using Monte Carlo Tree Search. ACM Trans. Recomm. Syst.

Chapter 3 - User-Cold Start and Item Recommendation using Reinforcement Learning Transformer: RLT4Rec : In this chapter, we propose RLT4Rec, a transformer-based model for providing recommendations. We investigate how RLT4Rec can be applied towards user-cold start learning, and also how it can continue to provide recommendations for long-term users. By leveraging the architecture of a transformer model, the RLT4Rec avoid the use of a separate *'state estimator'* and learns its own internal state representation about the user, using the past sequence of user interactions. Through controlled experiments and synthetic datasets, we attempt to understand some of the learned features of RLT4Rec.

To summarize the contributions in this Chapter:

1. We propose a transformer-based approach to provide recommendations to both new users and established users in the same framework. This approach is much more scalable than the MCTS while offering better cold-start performance.

2. We use experimental techniques to understand the learned user-representations within the model: which is a novel concept not seen in Recommender Systems research.

3. We provide the implementation of RLT4Rec, which also includes the simulation environment. Researchers can use the evaluation environment to conduct

controlled experiments to analyse Recommender performance under synthetic user conditions.

This chapter is based on the following publications:

- Dilina Chandika Rajapakse and Douglas Leith. 2024. RLT4Rec: Reinforcement Learning Transformer for User Cold Start and Item Recommendation. ACM Trans. Recomm. Syst. (Under Review)
- Dilina Chandika Rajapakse and Douglas Leith. 2024. A Good State Estimator Can Yield A Simple Recommender: A Reinforcement Learning Perspective. 4rd International Workshop on Online and Adaptive Recommender Systems (OARS).

Chapter 4 - Reassessing the Effectiveness of Reinforcement Learning based Recommender Systems for Sequential Recommendation: Reinforcement Learning based Recommender Systems have been gaining popularity in sequential recommendation tasks. When conducting research on reinforcement learning based recommender systems, we found that the majority of the existing literature cannot be reproduced. At the same time, we noticed several inconsistencies in the claims and experimental setups used in this literature. This motivated us to conduct a study on the existing RL-based recommender systems in sequential recommendations and their effectiveness, compared to traditional supervised-learning approaches.

To summarize the contributions in this Chapter:

1. Detailed assessment and comparison of RL-based RS against non-RL session-based recommender algorithms, conducted under several evaluation protocols. This provides a fair reference for researchers when comparing different approaches.
2. We provide the resources to reproduce and benchmark selected session-based recommendation algorithms with common evaluation protocols. These include data pre-processing techniques, official model implementations and evaluation methods, that can be easily utilized in future research or independent studies.

This chapter is an outcome of a research collaboration with Prof. Dietmar at the University of Klagenfurt, and is based on the following publication:

- Dilina Chandika Rajapakse and Jannach Dietmar. 2025. Reassessing the Effectiveness of Reinforcement Learning based Recommender Systems for Sequential Recommendation. SIGIR. (Under Review)

Chapter 2

User Cold-Start Learning in Recommender Systems using Monte Carlo Tree Search

In our first research chapter, we focus on user-cold start learning for new users in a system. We study the use of Monte Carlo Tree Search (MCTS) - a model-based reinforcement learning technique to learn the preferences of new users. The MCTS-based recommender treats the user cold-start task similar to a single-player game, where it dynamically select the sequence of items presented to a new user with the goal of accurately identifying the new user's preferences.

We find that our MCTS-based cold-start approach is able to consistently quickly identify the preferences of a user with significantly higher accuracy than with either a decision-tree or a bandit-based approach i.e the learning performance is fundamentally superior. This boost in recommender accuracy is achieved in a computationally lightweight fashion. The MCTS approach is flexible in the sense that it can readily extended to incorporate different types of user feedback including explicit ratings, ranked comparisons and missing not at random data.

Our MCTS-based cold start implementation and data is available on github at https://github.com/dilina-r/mcts-rec.

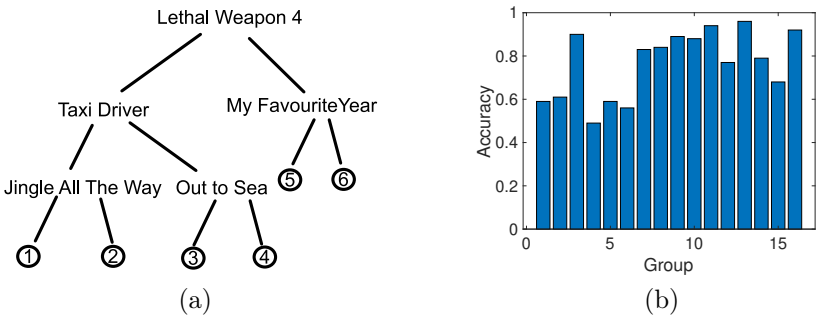
2.1 Introduction

In this chapter we consider the cold-start task for new users of a recommender system, which remains a core challenge. When a new user joins the system it initially has no knowledge of the preferences of the user and so would like to quickly learn these[1]. In a cold-start scenario, non-personalised methods such as recommending popular, recent or best-rated items can be used. However, these items are poor when it comes to providing insights into a user's preferences, and are observed to require more item-interactions to learn about these preferences. Therefore the recommender system initially starts in an "exploration" phase where the first few items that it asks the new user to rate are chosen with the aim of discovering the user's preferences. We also take into account that additional side information about the users/items are not always available, therefore we begin by focusing on the simplest setup where a user explicitly rates items presented to them, e.g. on a 1-5 scale or binary like/dislike feedback, and the aim of the recommender system is to predict other items that the user may like. We also consider that a real recommender can have various types of information gathering and user-feedback mechanisms. We then show that our approach can be extended to other types of feedback, in particular, to ranked comparison feedback (where the user is presented with a list of items and asked which they like best) and so-called missing not at random (MNAR) feedback (where a user may choose not to rate an item, e.g. due to lack of knowledge, and this may itself be informative of user preferences).

One common approach to the new user cold-start task is to take ratings already collected from a population of users, use these to cluster users into groups and then train a decision-tree to learn a mapping from item ratings to the user group, see for example Figure 2.1(a). When a new user joins the system this decision-tree is used to decide which items the user is initially asked to rate and in this way the group to which the user belongs is initially estimated. Once the group is estimated, the system recommends items liked by members of that group e.g. using matrix factorisation (MF) or another collaborative filtering approach.

However, typically users clustered in the same group do not give identical ratings to an item. Rather there is a spread of ratings, and this intra-cluster variability between users can be thought of as adding noise to the ratings. Unfortunately, decision trees can easily make mistakes in the face of such noise. For example, Figure 2.1(b) shows the measured decision-tree accuracy for Netflix data clustered into 16 groups (see later for more details). It can be seen that the accuracy is as low as 50-60% for a number of groups.

[1]The system may have some general context regarding the user, such as the country/city they are located in, user gender and demographics *etc*, in which case learning is conditioned on this but the fundamental cold start task otherwise remains unchanged.



(a) (b)

Figure 2.1: (a) Illustrating a movie recommender decision-tree (adapted from (Zhou et al., 2011)), (b) Decision-tree accuracy for Netflix data (16 groups).

The user cold-start task can be viewed as a form of single-player game. In a single-player game a sequence of moves is selected with the aim of maximising the reward or score generated by an environment (which may have some randomness). Supposing t moves have already been made, then lookahead to the next sequence of d moves can be represented by a path in a tree of depth d. Selecting the next move then involves exploring this tree to find a good future sequence of moves and then making the first move from that sequence, and the process then repeats starting from move $t + 1$.

We can directly map this to user cold-start as follows. A move consists of presenting a user with an item and observing their rating. After t moves we have presented a user with t items and observed t ratings. The next sequence of d moves consists of a path in a tree of depth d where each node is an item. The score of a sequence is the estimated probability of learning the correct user group by presenting that sequence of items to the user (we discuss the details of this calculation below). The first item of the sequence of d items with the highest score is then presented to the user, and the process then repeats. Since we have a budget of t_{max} items, the maximum lookahead d at step t is equal to $t_{max} - t$.

Observe that selecting the next move/item-to-present involves searching a lookahead tree of depth j. Monte Carlo Tree Search (MCTS) replaces exhaustive exploration of this tree with targetted exploration of only the most promising paths. This allows deeper tree exploration and better solutions to be found, even when the tree branching factor (the number of possible next moves/items) is large. MCTS has now largely replaced A/B trees in computer game playing and has contributed to breakthrough performance in Go (e.g. see AlphaGo (Silver et al., 2016)) and Chess (e.g. see (Silver et al., 2018)).

We find that an MCTS-based approach to cold-start is able to achieve extremely fast learning of user preferences. We demonstrate that the group of a user is consistently quickly identified with significantly higher accuracy than with either a

decision-tree or a state-of-the-art bandit-based approach without incurring higher regret[2] i.e the learning performance is fundamentally superior to that of the state of the art. This boost in recommender accuracy is achieved in a computationally lightweight fashion, with our MCTS-based implementation able to perform over 30+ recommendations per second using a single CPU core for the Netflix dataset with 8 distinct user groups. It is trivially parallelisable and so linearly scalable with the number of CPU cores.

2.2 Related Work

For a comprehensive survey of solutions to cold-start see (Elahi et al., 2016, 2018; H. Yuan & Hernandez, 2023). Collaborative filtering based recommenders are known to handle the cold-start problem poorly when a new user with no prior ratings or item-interactions arrives. Passive approaches include recommending popular items, use of item-based recommendation (once a user starts rating items an item-based approach is used to recommend similar items), transfer learning from another recommender system previously used by a user, and asking new users to rate a fixed list of items. Examples of early work on active learning include IGCN (information gain through clustered neighbours) which uses a decision tree with user clusters as leaves (Rashid et al., 2008) and the ternary decision-tree approach of (Golbandi et al., 2011). More recently, (Amatriain et al., 2009) uses representative items i.e. after completing the rating matrix R, k columns of R are selected, the ratings of the other items are represented as a linear combination of these and during cold start a new user is asked to rate these representative items. This approach is extended to use a decision-tree approach by (Shi et al., 2017). In (Zhou et al., 2011) a matrix factorization approach is proposed whereby a decision-tree is trained to map from item ratings to the latent feature vector for a user.

During user-cold start, one method of providing personalised recommendations is using content-based techniques (Lops et al., 2011) which incorporate additional attributes and side information of the users and items. Often content-based techniques are combined with collaborative filtering and deep learning methods in hybrid approaches (R. Chen et al., 2018; Choi, 2014; He et al., 2017; Volkovs et al., 2017) to tackle cold start. Graph Neural Network based methods (Cai et al., 2023; Hao et al., 2021; Zhang et al., 2022) have also shown notable cold start performance in recent years. They are able to capture the relationship between users, items and their attributes, together with the aggregated information from the neighbours.

The use of multi-arm bandits (MABs) for user cold-start has also received at-

[2]regret is the difference between the rating of predicted item and optimal item i.e: item that yields higher rating

tention. In (Felício et al., 2017) after completing the rating matrix R its rows are clustered and the average rating vector for each cluster is used as a representative user. During cold start, an MAB is used to select the average ratings vector to use and the user is asked to rate the next highest item in the vector. In (Canamares et al., 2019) a MAB is used to select between recommender strategies, typically recommending popular items initially for a new user and later switching to a k-Nearest Neighbor (kNN) or matrix factorisation (MF) model. Note that naive application of standard bandit algorithms to the cold-start task leads to poor performance. If we think of each recommender system item as an arm of a MAB then we run into the difficulty that (i) there are many arms and so learning is slow and (ii) repeated pulls of the same arm tend to be highly correlated. One remedy is to associate an arm with each group rather than each item (Felício et al., 2017). For each group the available items are sorted in descending order of their predicted rating by users in that group. Pulling the arm for a group then corresponds to asking the user to rate the next item from this sorted list, i.e. the unrated item predicted to have the highest rating for members of the group. While this greatly reduces the number of arms in the MAB, the learning rate remains very slow (Shams et al., 2021). This is because items rated highly by members of one group tend to also be rated highly by members of at least some of the other groups, and so the user ratings for these items do not serve to strongly distinguish between groups and so allow rapid learning. To address this, (Shams et al., 2021) identify so-called distinguisher items that tend to have distinct ratings by users in different pairs of groups. Using these they propose a Cluster-based Bandit algorithm that we use as a baseline for comparison in our research.

Contextual bandits have also received attention for user cold-start. These make use of contextual information, e.g. the user's location, gender, and demographics. In this chapter, we assume that such contextual information is absent but using similar techniques to contextual bandits our approach can be readily extended to take advantage of it when it is available.

Monte Carlo Tree Search was introduced by (Coulom, 2006) and quickly adopted within the two-player game community. See (Browne et al., 2012) for a survey of MCTS methods developed in the first 5 years after its introduction. MCTS was used by AlphaGo (Silver et al., 2016) to defeat the world champion in the game of Go, and also in the later AlphaZero (Silver et al., 2018) game-playing engine. Use of MCTS in single-player games was introduced in (Schadd et al., 2008, 2012). For more recent work, see for example (Crippa et al., 2022) and references therein.

Offline evaluations are often used in RS research (including cold-start) as a standard method of comparing different techniques. This allows for controlled experiments using historical data, providing efficient evaluation and flexibility for algo-

rithm development using existing real-world datasets. Offline testing is said to be reflective of online performance, but the correlation is shown to be weak (Jannach & Jugovac, 2019). In an online method, the algorithm is evaluated while interacting with real users and in real-time. This is the best, but only a few researchers have access to these live systems and can impose limitations on reproducibility. At the same time, simulated testing environments have recently gained popularity, where synthetic datasets can be generated to mimic user-item interactions. These can be simply derived from pre-defined behaviours or learned from real-life datasets to emulate them. For example (F. Liu et al., 2018) perform simulated online-testing by generating user-ratings for unrated items using a pre-trained PMF (Mnih & Salakhutdinov, 2007) model. Similarly, BLC-MF(Checco et al., 2017) can be used to generate synthetic ratings to mimic the user population in real datasets. We can find simulations to be widely incorporated in reinforcement learning based RS research (see (Afsar et al., 2022) for detailed survey on RLRS). The use of simulations in user-cold start is scarce.

Explicit feedback (i.e: user item-ratings) can provide valuable information to learn about a user. However, recommenders should also consider a wider variety of *implicit* feedback, including ranked comparison and missing not-at-random (MNAR) feedback. Ranked comparison is a popular approach for eliciting user feedback in which a user presented with a ranked list of items and clicks on the item that they most prefer. In its simplest form, a user is presented with two items and picks the best one. Pairwise approaches have, for example, been considered by (J. Liu et al., 2022; Pan & Chen, 2013; Rendle et al., 2009) in the learning techniques. When asked to rate an item or to choose the best item from a list, a user may choose not to respond. This non-response can reflect lack of knowledge of the items or some other aspect of user preference/behaviour, in which case it may be informative i.e. provide a type of implicit feedback. Such MNAR feedback (and associated ratings bias) has, for example, been considered by (J. Chen et al., 2021; Dalvi et al., 2013; Hernández-Lobato et al., 2014; Sindhwani et al., 2010; X. Wang et al., 2019). While ranked comparison and MNAR feedback has been the subject of much interest in recommender systems generally, its use during cold start has, to the best of our knowledge, largely been neglected to date.

2.3 Monte Carlo Tree Search For User Cold-Start

2.3.1 Preliminaries

We have a set $\mathcal{G}$ of user groups. Each user belongs to one group $g \in \mathcal{G}$. We also have a set of questions $\mathcal{V}$ e.g. a set of items the user is asked to rank. Given a new

user, our task is to quickly learn which group they belong to by asking the user to answer t_{max} questions from $\mathcal{V}$, using the fact that the distribution of item ratings (and so question answers) varies depending on the user's group.

Let $p_g^{(t)}$, $g \in \mathcal{G}$ be an estimate of the probability that the user belongs to group g given the user has answered questions $\mathcal{V}^{(t)}$, with $\sum_{g\in\mathcal{G}} p_g^{(t)} = 1$ and $0 \leq p_g^{(t)} \leq 1$. Initially, for a new user $t = 0$, $\mathcal{V}^{(0)}$ is the empty set and the probabilities $p_g^{(t)}$, $g \in \mathcal{G}$ can be initialised to the uniform distribution $p_g^{(0)} = 1/|\mathcal{G}|$, or alternatively to a distribution derived from population data.

Our MCTS approach is agnostic to how the probabilities $p_g^{(t)}$ are calculated, so long as they capture user feedback. In our examples, we will consider the following three approaches, and for concreteness assume that user ratings are Gaussian although the extension to more general setups is straightforward.

Explicit Item Ratings

A user is presented with an item and responds with a numerical rating. For users belonging to group g the rating $R(v)$ of item v is independent and identically distributed (i.i.d.) gaussian with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. Denoting the user's group by random variable G we then have that

$$p(R(v) = r|G = g) = (1/\sqrt{2\pi}\sigma(g, v))e^{-(r-\mu(g,v))^2/2\sigma^2(g,v)}$$

and for observed sequence $D^{(t)}$ of ratings $R(v_1), R(v_2), \ldots, R(v_t)$ for items $\mathcal{V}^{(t)} = \{v_1, v_2, \ldots, v_t\}$ it follows that

$$p(D^{(t)}|G = g) = \gamma^{(t)}(g)e^{-L^{(t)}(g)}$$

where $L^{(t)}(g) := \sum_{i=1}^{t}(R(v_i)-\mu(g, v_i))^2/2\sigma^2(g, v_i)$, $\gamma^{(t)}(g)) := 1/(2\pi)^{t/2}\times 1/\Pi_{i=1}^{t}\sigma(g, v_i)$. By Bayes' rule,

$$p_g^{(t)} = p(G = g|D^{(t)}) = \frac{p(D^{(t)}|G = g)p(G = g)}{p(D^{(t)})} = \frac{p(D^{(t)}|G = g)p(G = g)}{\sum_{h\in\mathcal{G}} p(D^{(t)}|G = h)p(G = h)}$$

with $p(D^{(t)}) = \sum_{h\in\mathcal{G}} p(D^{(t)}|G = h)p(G = h)$. Assuming uniform prior $p_g^{(0)} = p(G = g) = 1/|\mathcal{G}|$ then

$$p_g^{(t)} = \frac{\gamma^{(t)}(g)e^{-L^{(t)}(g)}}{\sum_{h\in\mathcal{G}} \gamma^{(t)}(h)e^{-L^{(t)}(h)}} \tag{2.1}$$

for $t = 1, 2, \ldots$.

Ranked Comparison

A user is presented with a list of items and responds by selecting the item that they like best. Similarly to above, we assume that for users belonging to group g the rating $R(v)$ of item v is i.i.d. gaussian with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. However, now we do not directly observe this rating. t Instead, for a pair of items v, w we observe whether $R(v) > R(w)$ or $R(v) < R(w)$. We have that

$$p(R(v) > R(w)|G = g) = p(R(v) - R(w) > 0|G = g)$$

and $R(v) - R(w)$ is gaussian with mean $\mu(g, v) - \mu(g, w)$ and variance $\sigma^2(g, v) + \sigma^2(g, w)$. Let Z be an indicator random variable that takes value 1 when $R(v) - R(w) > 0$ and 0 otherwise. Then,

$$p(Z = 1|G = g) = 1 - \text{cdf}\left(-\frac{\mu(g, v) - \mu(g, w)}{\sqrt{\sigma^2(g, v) + \sigma^2(g, w)}}\right)$$

where cdf denotes the cumulative distribution function of a gaussian random variable with mean zero and variance 1. For observed sequence $Z^{(t)} = \{Z_1, Z_2, \ldots, Z_t\}$ for pairs of items $\mathcal{V}^{(t)} = \{(v_1, w_1), \ldots, (v_t, w_t)\}$ it follows that

$$p(Z^{(t)}|G = g) = \prod_{i=1}^{t} p(Z_i|G = g)$$

Using the Bayes rule,

$$p(G = g|Z^{(t)}) = \frac{p(Z^{(t)}|G = g)p(G = g)}{p(Z^{(t)})}$$

with $p(Z^{(t)}) = \sum_{h \in G} p(Z^{(t)}|G = h)p(G = h)$ and assuming prior probabilities $p(G = h) = \frac{1}{|G|}$ it follows that,

$$p(G = g|Z^{(t)}) = \frac{p(Z^{(t)}|G = g)}{\sum_{h \in G} p(Z^{(t)}|G = h)} \tag{2.2}$$

While the above is for pairs of items, the generalisation to lists with more than two items is immediate.

Explicit Item Ratings With Missing Data

A user is presented with an item and responds either with a numerical rating or by providing no feedback (e.g. skips past the presented item). Let indicator random variable Y take value 1 when a user in group g provides a rating and 0 otherwise.

As above, the rating $R(v)$ of item v is i.i.d. gaussian with mean $\mu(g,v)$ and variance $\sigma^2(g,v)$.
When rating $R(v)$ is provided,

$$p(R(v)=r|G=g)=(1/\sqrt{2\pi}\sigma(g,v))e^{-(r-\mu(g,v))^2/2\sigma^2(g,v)}P(Y=1|G=g)$$

When no rating is provided, $p(\text{no rating}|G=g)=P(Y=0|G=g)$.

For observed sequence $Y^{(t)}$ of pairs $(Y_1,R(v_1)),(Y_2,R(v_2)),\ldots,(Y_2,R(v_t))$ for items $\mathcal{V}^{(t)}=\{v_1,v_2,\ldots,v_t\}$, then

$$p(G=g|Y^{(t)})=\frac{p(Y^{(t)}|G=g)}{\sum_{h\in G}p(Y^{(t)}|G=h)} \tag{2.3}$$

assuming uniform prior $p_g^{(0)}=p(G=g)=1/|\mathcal{G}|$.

2.3.2 Efficiently Searching The Lookahead Tree

To streamline the presentation we will use the explicit ratings setup as a running example when discussing our MCTS approach in the next sections. When displaying an item to a user is mentioned, then with ranked comparison feedback substitute display of a list of items (and each node in the lookahead tree is now a list). When a user rating response is mentioned, then with ranked comparison substitute selection of the best item, and when missing data is possible the response may either be a rating value or no rating. These user responses can be used to update the estimated probability $p_g^{(t)}$ that the user belongs to group g at step t of the cold start process as discussed above.

Suppose at step t a new user has rated items $\mathcal{V}^{(t)}$. The lookahead tree at step t embodies all item sequences of length $d=t_{max}-t$ where t_{max} is the total number of cold start items to be presented to a new user, e.g see Figure 2.2. Note that we only ask a user to rate a given item once since repeated ratings of the same item tend to be highly correlated[3]. To select the next item to present to the user our MCTS-based approach proceeds by repeatedly executing the following actions: (i) select a sample lookahead path $\mathcal{S}=\{s_{t+1},\ldots,s_{t_{max}}\}$ of $d=t_{max}-t$ items from the tree, (ii) draw a random user group $\hat{g}$ according to probability distribution $\{p_g^{(t)},g\in\mathcal{G}\}$, (iii) draw synthetic ratings $R(s_{t+1}),\ldots,R(s_{t_{max}})$ for a user from group $\hat{g}$, (iv) use $\mathcal{V}^{(t)}$ and synthetic ratings $R(s_{t+1}),\ldots,R(s_{t_{max}})$ to estimate the group probabilities $p_g^{(t_{max})}$, (v) if $\hat{g}$ is the group with highest probability generate reward +1 (the user

[3]Measurement studies indicate that when people are repeatedly asked to rate the same item on a scale of 1-5 then if they rate 1 or 5 they tend to consistently stick with that rating although when they rate 3 or 4 they may change their rating back and forth between 3 and 4. That is, lumping ratings of 3-4 together in a single bucket these previous studies indicate that a user's rating of an item tends to be consistent.

group is correctly identified) else generate reward 0, (vi) backpropagate this reward along path $\mathcal{S}$ in the lookahead tree. We give more details on these steps below.

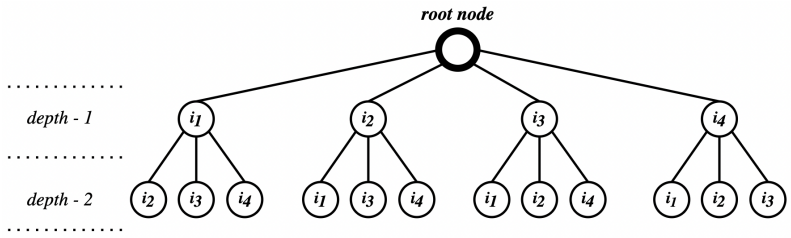


Figure 2.2: Example of lookahead tree with $|\mathcal{V}| = 4$, $d = 2$

In principle, in step (i) we would like to select a different path at every turn until all paths in the lookahead tree have been visited. We can then select the path with the highest reward, present the first item in that path to the user, observe the user's rating, update t to $t + 1$, $\mathcal{V}^{(t)}$ to $\mathcal{V}^{(t+1)}$ and repeat the MCTS steps. However, the lookahead tree is generally far too large for an exhaustive search over all paths to be feasible. To see this observe that in a lookahead tree of depth d at time $t = 0$ there is a root node with $|\mathcal{V}|$ children (corresponding to each possible item in set $\mathcal{V}$), and each child in turn has $|\mathcal{V}| - 1$ children and so on. The tree therefore has $|\mathcal{V}|!/d!(|\mathcal{V}| - d)!$ leaf nodes and when, for example, $|\mathcal{V}| = 1000$ and $d = 5$ then the tree has around 10^{12} leaf nodes. Rather than trying to carry out an exhaustive search of the lookahead tree, instead in step (iii) we try to identify good paths that are likely to generate high reward and focus on exploring these.

Finding Good Paths

Given that the full lookahead tree expands factorially and is thus too large for exhaustive search. We instead aim to identify and expand only the most promising branches, which we refer to as "good paths". These are sequences of items that are likely to yield higher rewards. Conversely, branches that generate lower rewards are visited few times and eventually abandoned, effectively pruning them from the search space as the algorithm converges on better paths.

Each node n in this subtree lies at the end of a path (i.e. sequence of items) $\mathcal{S}(n)$ that leads from the tree root to the node. Associated with each node n in the tree is: (i) an item $s(n)$ (namely, the last item in path $\mathcal{S}(n)$), (ii) a count $N(n)$ of the number of times that $s(n)$ has been included in the MCTS lookahead set $\mathcal{S}$ and (iii) a reward $Q(n)$ which is the sum of the rewards generated by lookahead sets $\mathcal{S}$ that include $s(n)$.

We construct this subtree (and in the process also generate sample lookahead paths) as follows. Initially create a tree consisting of a root node and $|\mathcal{V}|$ child nodes i.e. each child node is associated with a different item in $\mathcal{V}$. Initialise counters

$N(\cdot) = 0$ and $Q(\cdot) = 0$ for these child nodes. Pick a child node n uniformly at random and set item $s_{t+1} = s(n)$. We need to extend this to obtain a sequence $\mathcal{S}$ of d items, and the simplest way to do this is just to pick $d-1$ items $s_{t+2}, \ldots, s_{t_{max}}$ uniformly at random from $\mathcal{V}$. Generate the reward for this path $\mathcal{S}$ and add this to counter $Q(n)$ associated with the selected child node, also increment the $N(n)$ counter associated with the child node. At the next turn, select a child node uniformly at random from the child nodes with $N(\cdot) = 0$ and repeat this process.

Once there are no child nodes with $N(\cdot) = 0$, pick a child node n with high reward, set item $s_{t+1} = s(n))$ and add $|\mathcal{V} \setminus \{s(n)\}|$ child nodes to this node. Select one of these children uniformly at random set this as item s_{t+2}. Extend sequence $\{s_{t+1}, s_{t+2}\}$ to obtain a sequence $\mathcal{S}$ of length d by selecting $d-2$ items uniformly at random. Generate the reward for this path $\mathcal{S}$ and add this to the counters $Q(\cdot)$ of the nodes corresponding to items s_{t+1} and s_{t+2}, also increment the $N(\cdot)$ counters associated with these nodes.

At the next turn, repeat this process. That is, traverse the current tree picking nodes with high rewards until either a leaf node is reached or a node with unvisited children. If a leaf node n is reached then expand it by adding $|\mathcal{V} \setminus \mathcal{S}(n)|$ child nodes, and pick one of these at random. If a node with unvisited children is reached, pick uniformly at random from the unvisited children. This yields a partial sequence of items, of length equal to the depth of the tree. This is then extended to a sequence $\mathcal{S}$ of length d by selecting the remaining items uniformly at random. Generate the reward for this path $\mathcal{S}$ and update the $Q(\cdot)$ and $N(\cdot)$ counters of the nodes in the tree on path $\mathcal{S}$.

Algorithm 1 gives pseudo-code for this process of growing a lookahead subtree containing paths that tend to generate high rewards.

Selecting a Node With High Reward

Once the tree has started to grow, the process described above requires selecting a sequence of child nodes with high rewards until a leaf node is reached. In doing this we would like to balance exploration (trying new items) and exploitation (selecting an existing item) and so we adopt an optimistic upper confidence bound (UCB) approach. Namely, we estimate the UCB value of a node n that is a child of parent node p as

$$UCB(n) = \underbrace{\frac{Q(n)}{N(n)}}_{exploit} + \underbrace{\sqrt{0.25\frac{\log(N(p))}{N(n)}}}_{explore}$$

The first term $Q(n)/N(n)$ is the average reward observed so far for sequences that include item $s(n)$ (i.e. node n). However, when the number of samples $N(n)$ on

Algorithm 1 MCTS For User Cold-Start

$\mathcal{V}^{(t)} \leftarrow \emptyset$
for $t = 0; t = t + 1; t \leq t_{max}$ **do**
  Initialise lookahead tree $\mathcal{T} \leftarrow$ root node
  for turn $i = 1; i = i + 1; i \leq max\_turns$ **do** ▷ Construct lookahead subtree
    $j = t + 1$
    $n_j \leftarrow best\_child(\mathcal{T}_{root})$; $s_j = s(n_j)$ ▷ Select the good path
    while (n_j != null) and ($j < t_{max}$) **do**
      $n_{j+1} \leftarrow best\_child(n_j)$; $s_{j+1} = s(n_{j+1})$
      $j \leftarrow j + 1$
    end while
    if (n_j is leaf node) and ($j < t_{max}$) **then** ▷ Expand
      Add $|\mathcal{V} \setminus \mathcal{S}(n_j)|$ children to n_j, initialise children's $Q(\cdot)$, $N(\cdot)$ to 0
      $n_{j+1} \leftarrow best\_child(n_j)$; $s_{j+1} = s(n_{j+1})$
      $j \leftarrow j + 1$
    end if
    while $j < t_{max}$ **do** ▷ Randomised rollout
      $s_{j+1} \leftarrow$ select random item
      $j \leftarrow j + 1$
    end while
    $Q \leftarrow calculate\_reward(\mathcal{V}^{(t)}, \{s_{t+1}, \ldots, s_{t_{max}}\})$ ▷ Generate reward
    for node n in path $\mathcal{S}$ **do** ▷ Backpropagate reward
      $N(n) \leftarrow N(n) + 1$; $Q(n) \leftarrow Q(n) + Q$
    end for
  end for
  $n^* \leftarrow \arg\max_{n \in children(\mathcal{T}_{root})} N(n)$ (break ties randomly) ▷ Pick the most visited child of tree root
  $v_{t+1} \leftarrow s(n^*)$ ▷ Ask the user to rate the corresponding item
  $\mathcal{V}^{(t+1)} \leftarrow \mathcal{V}^{(t)} \cup \{v_{t+1}\}$
end for

which this average reward is based is small then the value may well be inaccurate. The second term aims to compensate for this. Intuitively, when $N(n)$ is small the second term is large, encouraging exploration of node n. As $N(n)$ grows, however, the second term becomes smaller and this captures the fact that the first term $Q(n)/N(n)$ is then more reliable. Another way to derive the expression for $UCB(n)$ is to note that the reward is a Bernoulli random variable taking the value 0 or 1. Applying Hoeffding's inequality, $P(\frac{Q(n)}{N(n)} \geq \mu + x) \leq e^{-2x^2 N(n)}$. Plugging in $x = \sqrt{0.25 \frac{\log(N(p))}{N(n)}}$ then $e^{-2x^2 N(n)} = e^{-0.5 \log(N(p))} = 1/\sqrt{N(p)}$. Hence, $UCB(n)$ can be thought of as the upper limit of a confidence interval which increases in power as the number of times $N(p)$ that the parent node is visited grows.

Algorithm 2 gives pseudo-code using this UCB approach to select a child node with a high reward.

Algorithm 2 best_child

Input: node p of lookahead tree $\mathcal{T}$
$C \leftarrow$children of p
if $C == \emptyset$ **then**
  return null ▷ No child nodes
else
  $C_{unvisited} \leftarrow \{n \in C : N(n) = 0\}$ ▷ Set of unvisited children of node p
  if $C_{unvisited} \neq \emptyset$ **then**
    return child selected uniformly at random from $C_{unvisited}$
  else
    return $\arg\max_{n \in C} \frac{Q(n)}{N(n)} + \sqrt{0.25 \frac{\log(N(p))}{N(n)}}$ (break ties randomly) ▷ UCB
  end if
end if

Algorithm 3 calculate_reward

Input: Item sequences $\mathcal{V}^{(t)}$, $\mathcal{S}$
Draw user group $\hat{g}$ according to probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$
Using $\hat{g}$, generate synthetic user rating $R(s)$ for each item $s \in \mathcal{S}$
Calculate $p_g^{(t_{max})}$, $g \in \mathcal{G}$ using equation (2.1)
if $\hat{g} \in \arg\max_{g \in G} p_g^{(t_{max})}$ **then**
  return $+1$ ▷ Estimated group matches "true" group $\hat{g}$
else
  return 0
end if

Calculating The Reward With Explicit Item Ratings

Given the items $V^{(t)} = \{v_1, \ldots, v_t\}$ already rated by a user, plus a sample path $\{s_{t+2}, \ldots, s_{t_{max}}\}$ from the lookahead tree, we need to calculate the reward associated with the sequence of items $\{v_1, \ldots, v_t, s_{t+2}, \ldots, s_{t_{max}}\}$. We have the user ratings for items $v_1, \ldots, v_t$ but we lack ratings for items $s_{t+2}, \ldots, s_{t_{max}}$.

We therefore adopt a Monte Carlo sampling approach. Namely, we select a user group $\hat{g}$ according to the probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$ i.e. according to our best estimate of the user group at step t. Initially, we are unsure of the user group and $\{p_g^{(0)}\}$ is the uniform distribution, but as the user rates items we hope that the distribution $\{p_g^{(t)}\}$, and so $\hat{g}$, will start to concentrate on the true group of the user. This concentration behaviour can be seen, for example, in Figure 2.3 which plots $\{p_g^{(t)}, g \in \mathcal{G}\}$ vs the number of items t rated by the user for the Netflix dataset with $|\mathcal{G}| = 8$ user groups. In this example the new user belongs to group $g = 4$ and $p_4^{(t)} \to 1$ as t increases while $p_g^{(t)} \to 0, g \neq 4$.

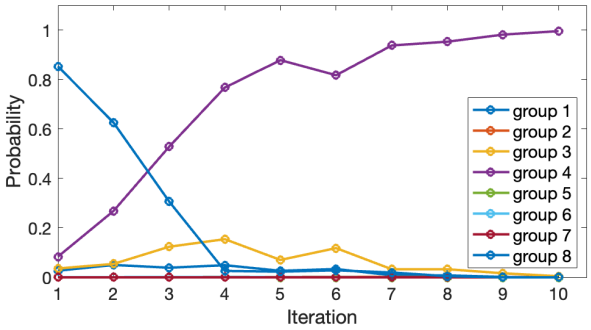


Figure 2.3: Example illustrating evolution of probabilities $p_g^{(t)}$, $g = 1, 2, \ldots, 8$ vs the number of items t rated by a new user in group $g = 4$. Netflix dataset, $|\mathcal{G}| = 8$ user groups, explicit item rating feedback. It can be seen that after rating $t = 3$ items the estimated probability $p_4^{(t)}$ of the user being in group 4 is higher than that of the other groups and $p_4^{(t)} \to 1$ as t increases.

Given user group $\hat{g}$, we can generate synthetic ratings $R(s_{t+2}), \ldots, R(s_{t_{max}})$ by making a draw from the multivariate Gaussian distribution with mean $\mu(\hat{g}, v)$ and variance $\sigma^2(\hat{g}, v)$, $v \in \mathcal{V}$ of ratings by users in group $\hat{g}$. Note that, alternatively, we could also draw ratings from the empirical distribution of ratings for a group (so relaxing the Gaussian assumption) and also by generating user ratings via a water-filling approach i.e. split the data into training and test data, pick a user from the test data and use their ratings, when we need a rating for an item that the user has not rated, pick a second user from the same group who has rated the item and merge the pair of user ratings. We found the performance of these setups to be very similar to simply drawing a new user from a Gaussian distribution.

With user ratings $R(v_1), \ldots, R(v_t)$ and synthetic ratings $R(s_{t+2}), \ldots, R(s_{t_{max}})$ in hand, equation (2.1) can now be used to calculate the estimated probability $\{p_g^{(t_{max})}\}$ when the user rates items $\{v_1, \ldots, v_t, s_{t+2}, \ldots, s_{t_{max}}\}$. If the "true" group $\hat{g}$ is the group with the highest estimated probability then the reward for the sequence is $+1$, else 0.

Averaging over multiple such samples, the average reward $Q(n)/N(n)$ will be

$$\frac{Q(n)}{N(n)} \approx \mathbb{E}[\text{Prob true group is correctly estimated} \mid \text{true group} = g, \\ \text{user ratings of items } V^{(t)}, \text{ next items are } s_{t+2}, \ldots, s_{t_{max}}]$$

where the expectation is over the groups $\mathcal{G}$.

Algorithm 3 gives pseudo-code for this Monte Carlo-based reward calculation.

Calculating The Reward With Ranked Comparison Feedback

The above approach can be directly extended to ranked comparison feedback. Now each node in the tree is a list of items. Given user group $\hat{g}$, we can generate a synthetic rating for each item in a list by making a draw from the multivariate Gaussian distribution of ratings by users in group $\hat{g}$. The item with the highest rating is then selected. Repeating this for each node in a path, equation (2.2) can now be used to calculate the estimated probability $\{p_g^{(t_{max})}\}$. If the "true" group $\hat{g}$ is the group with the highest estimated probability then the reward for the sequence is $+1$, else 0.

Algorithm 4 gives pseudo-code for this ranked comparison Monte Carlo reward calculation. The pseudo-code is for pairs of items but can be directly generalised to larger lists of items.

Algorithm 4 calculate_reward

Input: Prior events $Z^{(t)}$, $\mathcal{S}$
Draw user group $\hat{g}$ according to probability distribution $\{p_g^{(t)}, g \in \mathcal{G}\}$
Using $\hat{g}$, generate synthetic user rating $R(v_i)$, $R(v_j)$ for each item pair $(v_i, w_i) \in \mathcal{S}$
Calculate $p_g^{(t_{max})}$, $g \in \mathcal{G}$ using equation (2.2)
if $\hat{g} \in \arg\max_{g \in G} p_g^{(t_{max})}$ **then**
    return $+1$
else
    return 0
end if

Calculating The Reward With Missing Data

When users can respond by not providing a rating, then given user group $\hat{g}$ at each stage we draw an indicator random variable that takes value 1 with probability $p(Y = 1|G = \hat{g})$, and otherwise takes the value 0. We now replace equation (2.2) by equation (2.3) when calculating $\{p_g^{(t_{max})}\}$.

Choosing The Next Item

A lookahead subtree consisting of sequences of items that tend to generate high rewards is constructed by executing the Monte Carlo search steps (i)-(vi) max_turns times, where max_turns is a hyperparameter. We need to select max_turns to be large enough that promising sequences of items are discovered and are visited sufficiently frequently that the estimated reward for the sequence is reasonably accurate.

In the lookahead subtree, the children of the tree root are the first items in the set of lookahead sequences. Using the lookahead subtree generated at step t we choose the next item v_{t+1} to ask the user to rate to be the child of the tree root

that has been most visited during the Monte Carlo search i.e. $v_{t+1} = s(n^*)$ where $n^* \in \arg\max_{n \in children(\mathcal{T}_{root})} N(n)$. While we might alternatively choose the child with the highest estimated reward $Q(n)/N(n)$, we found that selecting the most frequently visited child tended to achieve slightly better performance.

2.3.3 Further Speedups

Best Distinguisher Items Are Enough

The lookahead tree expands by a factor of roughly $|\mathcal{V}|$ at each level. Even with the Monte Carlo search approach described above the computational and memory burden can therefore quickly become substantial for large $|\mathcal{V}|$. Following (Shams et al., 2021), we note that some items are more effective than others for distinguishing between groups. For example, popular items rated highly by members of one group can tend to also be rated highly by members of at least some of the other groups, and so the user ratings for these items do not serve to strongly distinguish between groups.

Intuitively, an item v helps to distinguish whether a user belongs to group g rather than group h when (i) the mean rating of v by users in group g is very different from that of users in group h i.e. $(\mu(g,v) - \mu(h,v))^2$ is large, and (ii) when the ratings tend to be consistent/reliable i.e. the variance $\sigma^2(g,v)$ is small. That is, we expect that

$$\Gamma_{g,h}(v) = \frac{(\mu(g,v) - \mu(h,v))^2}{\sigma^2(g,v)}$$

is a measure of the ability of item v to distinguish group g from group h i.e. the larger $\Gamma_{g,h}(v)$ the better item v is at distinguishing group g from group h.

Another way to arrive at the same conclusion is to assume that for users belonging to group g the rating $R(v)$ of item v is i.i.d. gaussian with mean $\mu(g,v)$ and variance $\sigma^2(g,v)$. Consider similarity measure

$$R_t(g,h) = \sum_{i=1}^{t} \frac{R(v_i) - \mu(h,v_i)}{\mu(g,v_i) - \mu(h,v_i)}$$

Suppose the user belongs to group g. We expect that the deviations $R(v_i) - \mu(g,v_i), i = 1,\ldots,t$ tend to fluctuate around 0, as otherwise there would be a consistent offset between the user's ratings and the group ratings in which case the user would better be assigned to a different group. Therefore $R_t(g,h) \to 1$ as $t \to \infty$ for $h \neq g$ and, similarly, $R_t(h,g) \to 0$ as $t \to \infty$ for $h \neq g$. Hence, by thresholding $R_t(g,h)$ we can identify the user group g. By standard concentration inequalities, $Prob(|R_t(g,h) - 1| > \epsilon) < 2e^{-\frac{\epsilon^2}{2}\sum_{i=1}^{t}\Gamma_{1,h}(v_i)}$ and and $Prob(|R_n(h,g) - 0| > \epsilon) < 2e^{-\frac{\epsilon^2}{2}\sum_{i=1}^{t}\Gamma_{h,g}(v_i)}$. That is, for $R_t(g,h)$ to converge quickly we want $\sum_{i=1}^{t}\Gamma_{g,h}(v_i)$ to

be large.

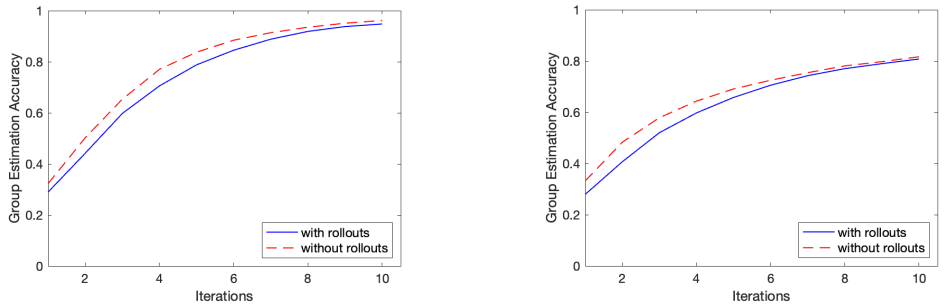
Using this observation, for each pair of groups $g, h \in \mathcal{G}$ we pick the t_{max} items v for which $\Gamma_{g,h}(v)$ is largest and add these to set $\hat{\mathcal{V}}$. Set $\hat{\mathcal{V}}$ is generally much smaller than set $\mathcal{V}$, e.g. Table 2.1 shows the size of $\hat{\mathcal{V}}$ for the standard Netflix dataset as the number of groups is varied. In our performance evaluation below we will select items to ask the user to rate from the subset $\hat{\mathcal{V}} \subset \mathcal{V}$ of distinguisher items rather than the full set of items $\mathcal{V}$.

	$\lvert\mathcal{V}\rvert$	$\lvert\hat{\mathcal{V}}\rvert$			
		4	**8**	**16**	**32**
Netflix	17,770	269	759	1,224	905
Jester	100	59	83	98	100

Table 2.1: Number of total items $|\mathcal{V}|$ vs number of best distinguisher items $|\hat{\mathcal{V}}|$ generated for Netflix and Jester datasets with 4,8,16 and 32 user groups.

No Rollouts

In Algorithm 1, after reaching a leaf node in the lookahead tree $\mathcal{T}$ of depth $j < t_{max}$ we select $j - t_{max}$ items at random to obtain a sample item sequence of length t_{max}. A *rollout* step of this sort is standard in MCTS. However, we also measured the performance of our MCTS-based approach when this rollout step was omitted i.e. a sample item sequence of length $j \leq t_{max}$ is generated from the lookahead tree.



(a) Netflix Dataset - 8 groups (b) Goodreads Dataset - 8 groups

Figure 2.4: Group estimation accuracy with and without rollout step for MCTS with explicit item rating feedback.

Figure 2.4 shows typical performance measurements for the Netflix and Goodreads datasets. This plots the accuracy with which the user group is estimated vs the number of items t_{max} that are rated by the user. Data is shown both with and without the rollout step. It can be seen that, perhaps somewhat surprisingly, the rollout step tends to degrade performance. What we suspect is happening here is that the rollout step is effectively just adding unhelpful noise since we may only sample a relatively small number of random item sequences from the full set of possible sequences

of length $j - t_{max}$, especially during the initial stages when j is small. Omitting the rollout step therefore offers the double advantage of reduced computation and improved performance.

Number Of Turns

The performance of the MCTS-based algorithm depends on the max_turns parameter that determines the number of Monte Carlo samples used to generate the lookahead subtree. This needs to be sufficiently large that promising sequences of items are discovered and are visited sufficiently frequently that the estimated reward for the sequence is reasonably accurate. Initially, we used a simple linear heuristic to $max\_turns = k \times |G| \times |\hat{\mathcal{V}}|$, where parameter k is varied depending on the dataset but it is typically around 200.

Dataset	*k*						
	1	*5*	*10*	*50*	*100*	*200*	*500*
Goodreads/8	0.564	0.608	0.615	0.659	0.679	0.710	0.703
Jester/8	0.485	0.491	0.530	0.524	0.536	0.545	0.551
Netflix/8	0.691	0.801	0.795	0.825	0.831	0.831	0.830

Table 2.2: Group estimation accuracy vs choice of max_turns parameter k for MCTS with explicit item rating feedback. Netflix, Goodreads and Jester datasets with $|\mathcal{G}| = 8$ groups, $t_{max} = 5$ items rated by new user

Table 2.2 shows the accuracy of the MCTS algorithm as the value of k is varied. Data is shown for the three datasets with $|\mathcal{G}| = 8$ groups and $t_{max} = 5$. It can be seen that the accuracy initially improves as k is increased. This is as expected since the lookahead subtree is being more thoroughly explored. However, once k gets to a value of about 200 the accuracy starts to level off, indicating that the lookahead tree is now sufficiently large and accurate.

The value of k where the accuracy levels off depends on depth $d = t_{max} - t$ of the lookahead tree ($d = 5$ in Table 2.2), a smaller value of k being admissible when d is smaller, and vice versa. In our tests we therefore used $k = (1.25 + (t_{max} - t)^2$.

2.4 Performance Evaluation

2.4.1 Evaluation Setup

Datasets. We evaluate performance on the standard Netflix dataset (480,189 people 17,770 movies, 104M ratings from 1–5), Movielens10M[4] (72,000 people 10,000

[4]https://grouplens.org/datasets/movielens/

movies, 10M ratings from 1–5), the Jester dataset [5] (73,421 people rating 100 jokes, 4.1M ratings from -10–10) and the Goodreads10K dataset [6] (53,424 people rating 10,000 books, 5.9M ratings from 1-5) (Wan & McAuley, 2018).

Clustering Users. We use training data to cluster users into groups and estimate the mean $\mu(g, v)$ and variance $\sigma(g, v)^2$ of the ratings by each group g for item v. We use the BLC matrix-factorization clustering algorithm (**BLC**) for this, although other clustering algorithms (such as k-means) might also be used. We vary the number of groups/clusters from 4 to 32 and report results for each.

Baseline Algorithms. When explicit rating feedback is available we compare the performance of the proposed MCTS-based approach against

- Popular (Pop): The most interacted items among all users in the population, are recommended to the user starting from the most interacted.
- Best-rated (Best): The items with the highest average rating across all user-groups are sorted and recommended to the user.
- Decision Tree (D-Tree): A Classification and Regression Trees (CART) optimised decision Tree
- Cluster Based Bandits (CBB): Multi-arm bandits based user-cold start recommendation algorithm of (Shams et al., 2021)

These are strong baselines with good performance for cold-start active learning. Decision-trees are often considered for use in cold-start while the recently proposed CBB-algorithm offers state-of-the-art performance (Shams et al., 2021).

When missing data is possible we use a variant of MCTS as a baseline, which again provides a strong benchmark for comparison. We also compare against the non-personalised popular method.

Modelling New Users. When evaluating explicit rating feedback we generate the item ratings of a new user from group g by making a single draw from the multivariate Gaussian distribution with mean $\mu(g, v)$ and variance $\sigma(g, v)^2$ for each item equal to that estimated from the training data. This has the advantage that we can easily generate large numbers of new users in a clean, reproducible manner. In addition, we also evaluated performance when drawing ratings from the empirical rating distribution of each group and also splitting the data into training and test data, picking a user from the test data and using their ratings. In the case where a user has not rated an item, we pick a second user from the same group who has rated the item and merge the ratings. We found the performance of these setups to be very similar to simply drawing a new user from a Gaussian distribution.

[5] https://eigentaste.berkeley.edu/dataset/
[6] https://mengtingwan.github.io/data/goodreads

When evaluating ranked comparison feedback we adopt a similar approach: draw multivariate Gaussian ratings for each item and when given a list of items select the item with the highest rating.

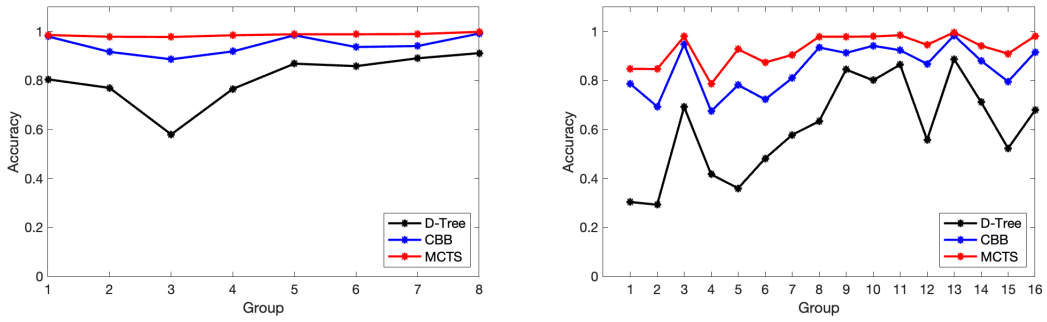
When evaluating ratings with missing data, unfortunately the datasets we used do not contain information on items presented to a user but not rated. For a given item we therefore calculate the number of times it is rated by users in each group and divide by the total number of ratings to get the relative frequency of rating by users in each group. We normalise these relative frequencies so that the minimum frequency is α, a design parameter that we will vary. Relative frequencies which are taken above 1 by this rescaling are clipped to 1. When $\alpha = 1$ then we recover the situation where there is no missing data and conversely a smaller α means a higher number of missing ratings. In our experiments, we vary the amount of missing data with the value of α set to 0.1, 0.2 and 0.5.

Performance Metrics. We report the Group Estimation Accuracy (also referred to as Estimation Accuracy), that is the accuracy with which the group of a new user is estimated (i.e. the fraction of times the correct group is estimated), vs the number of items rated by a new user. Statistics are calculated over 1000 new users per group.

Hardware. Tests were carried out on an 8-Core Intel i7-9700 CPU @ 3.00GHz, with 8GB RAM. Computational performance was measured using only a single core of the CPU.

2.4.2 Results With Explicit Rating Feedback

Figure 2.6 shows measurements of the mean accuracy vs the number of items rated by a new user for the Netflix, Goodreads and Jester datasets with users clustered in 4 and 16 groups. Data is shown when using popular items (pop), best-rated items (best), decision tree (D-Tree), the CBB algorithm of (Shams et al., 2021) and our proposed MCTS-based approach. To calculate the mean accuracy, 1000 new users are generated for each group and the accuracy averaged over the users and groups i.e. averaged over 4,000 users for 4 groups and 16,000 for 16 groups. It can be seen that the MCTS approach uniformly achieves a higher accuracy than all of the baselines, including the decision-tree and CBB approaches, for a given number of items rated.



(a) 8 groups (b) 16 groups

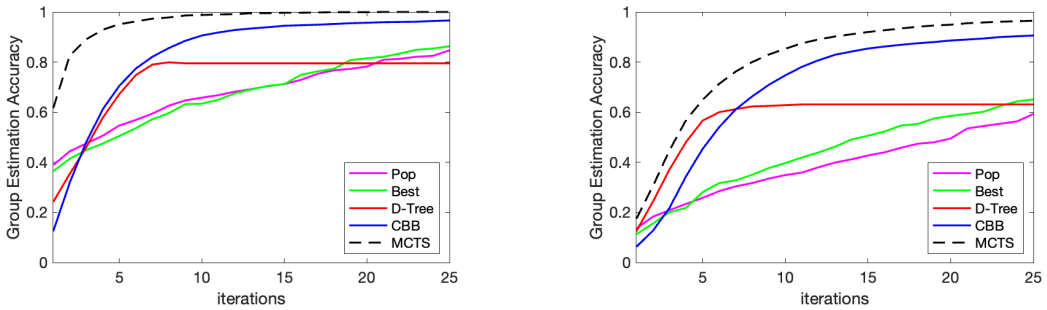
Figure 2.5: Measured per-group accuracy of the decision tree (D-Tree), CBB and MCTS-based approaches. Netflix dataset with 8 and 16 groups for $t_{max} = 15$ items rated by new users.

dataset/#groups	*algorithm*	*iterations*				
		1	*2*	*3*	*4*	*5*
Netflix/16	D-tree	0.359	0.194	0.196	0.196	0.196
	MCTS	**0.185**	**0.105**	**0.063**	**0.041**	**0.027**
Goodreads/16	D-tree	0.341	0.188	0.202	0.202	0.202
	MCTS	**0.216**	**0.173**	**0.137**	**0.113**	**0.094**
Jester/16	D-tree	**0.150**	0.146	0.154	0.154	0.154
	MCTS	0.182	0.138	0.114	0.092	0.076
Movielens/16	D-tree	0.379	0.202	0.202	0.205	0.205
	MCTS	**0.244**	**0.185**	**0.134**	**0.105**	**0.091**

Table 2.3: Standard deviation of the measured per-group accuracy for the decision tree (D-tree) and MCTS approaches, on Netflix, Jester, Goodreads and Movielens datasets with 16 groups.

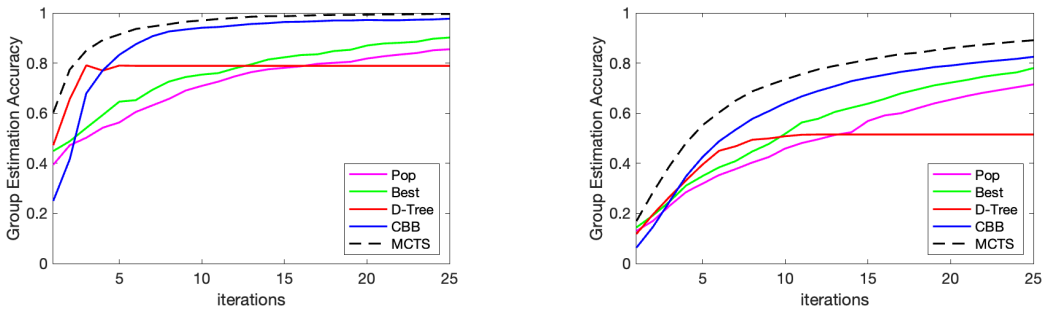
Figure 2.5 shows typical (i.e. representative of the full data) measurements of the per-group accuracy of the D-tree, CBB and MCTS approaches. It can seen that the MCTS approach consistently achieves higher accuracy for every group. The variation in the accuracy across groups is also lower, particularly in comparison to that of the decision-tree approach e.g. it can be seen in Figure 2.5(b) that the accuracy can range from about 30% to about 90% when using the decision-tree. Table 2.3 shows the standard deviations of the per-group accuracies vs the number of items rated by a new user for the Netflix, Goodreads and Jester datasets with 16 groups. Once again, it can be seen that the variation in accuracy across groups is significantly lower with the MCTS approach.

Tables 2.4–2.7 summarise the measured group estimation accuracy vs the number of items rated by a new user and the number of user groups. Data is shown for 5–25 items and 4–32 groups and for the Netflix, Goodreads and Jester datasets. It can be seen that the MCTS-based approach achieves uniformly superior performance to all other methods, including decision-tree and CBB approaches i.e. better performance



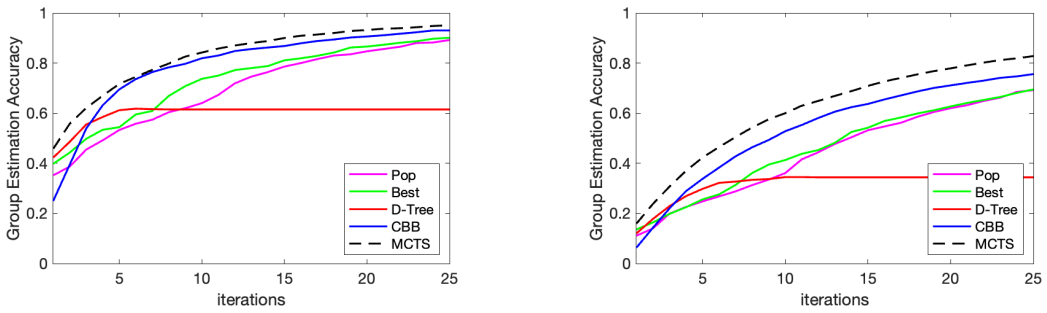
(a) Netflix 4 Groups

(b) Netflix 16 Groups



(c) Goodreads 4 Groups

(d) Goodreads 16 Groups



(e) Jester 4 Groups

(f) Jester 16 Groups

Figure 2.6: Measured mean group-estimation accuracy vs number of items rated by a new user (iterations), up to $t = 25$ item recommendations. Data is shown for the Popular (Pop), Best-rated (Best), Decision tree (D-tree), CBB and MCTS-based approaches for the Netflix, Goodreads and Jester datasets with 4 and 16 groups

regardless of the number of items the user rates, the number of user groups and the dataset used. The performance improvement is often considerable. For example, with 4 groups and asking the user to rate 5 items the MCTS accuracy on the Netflix dataset is 0.951 vs a decision-tree accuracy of 0.672 and a CBB accuracy of 0.705. Similarly, for 16 groups and the user rating 15 items the accuracies are 0.920, 0.631 and 0.854 for MCTS, D-tree and CBB respectively.

Since the performance of the MCTS approach dominates that of the other algorithms, this performance data indicates that it should always be used in preference to them if higher accuracy is desired.

Table 2.9 shows measurements of the average time taken to recommend the next

item to a new user, using the two methods Cluster Based Bandits (CBB) and our MCTS. For the MCTS this is the time taken at step t to construct a lookahead subtree containing item sequences that tend to generate high rewards, this subtree is then used to select the next item to ask a new user to rate. The data shown makes use of a single CPU core. The MCTS algorithm is massively parallelisable in the sense that computations for different new users can be trivially run in parallel and so is linearly scalable with the number of CPU cores. The time taken depends on the lookahead tree expansion factor (i.e. the size of the set of items that a user can be asked to rate), hence why the time is somewhat longer for the Netflix dataset than for Goodreads. It also depends on the MCTS max_turns parameter that determines the number of Monte Carlo samples used to generate the lookahead subtree, shown in Table 2.9. Even though the MCTS is observed to be capable of providing fast recommendations with a higher accuracy, we note that the CBB approach is computationally faster.

In summary, for 8 groups the MCTS-based approach takes around 30ms to recommend an item for a new user to rate, allowing over 30+ recommendations per second using a single commodity CPU core. Performance scales linearly with the number of CPU cores.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	Pop	0.547	0.658	0.712	0.782	0.847
	Best	0.505	0.634	0.712	0.815	0.863
	D-Tree	0.672	0.795	0.795	0.795	0.795
	CBB	0.705	0.906	0.945	0.957	0.966
	MCTS	**0.951**	**0.988**	**0.997**	**0.999**	**1.000**
8	Pop	0.410	0.508	0.601	0.672	0.756
	Best	0.393	0.565	0.648	0.728	0.770
	D-Tree	0.674	0.780	0.780	0.780	0.780
	CBB	0.701	0.903	0.942	0.955	0.964
	MCTS	**0.838**	**0.962**	**0.987**	**0.994**	**0.997**
16	Pop	0.258	0.349	0.427	0.495	0.593
	Best	0.281	0.397	0.506	0.585	0.651
	D-Tree	0.566	0.628	0.631	0.631	0.631
	CBB	0.453	0.747	0.854	0.886	0.906
	MCTS	**0.649**	**0.853**	**0.920**	**0.949**	**0.965**
32	Pop	0.173	0.264	0.338	0.404	0.506
	Best	0.210	0.341	0.437	0.506	0.572
	D-Tree	0.327	0.414	0.425	0.423	0.423
	CBB	0.272	0.517	0.651	0.715	0.755
	MCTS	**0.437**	**0.642**	**0.739**	**0.797**	**0.836**

Table 2.4: Mean group-estimation accuracy vs the number of items rated by a new user (iterations) for Netflix dataset with 4,8,16 and 32 user groups.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	Pop	0.563	0.709	0.781	0.818	0.855
	Best	0.646	0.754	0.823	0.870	0.902
	D-Tree	0.790	0.789	0.789	0.789	0.789
	CBB	0.833	0.941	0.964	0.972	0.977
	MCTS	**0.914**	**0.970**	**0.987**	**0.993**	**0.996**
8	Pop	0.486	0.627	0.715	0.787	0.816
	Best	0.564	0.692	0.777	0.829	0.869
	D-Tree	0.582	0.596	0.594	0.594	0.594
	CBB	0.593	0.774	0.817	0.851	0.874
	MCTS	**0.691**	**0.817**	**0.870**	**0.903**	**0.928**
16	Pop	0.319	0.459	0.569	0.654	0.715
	Best	0.349	0.518	0.638	0.722	0.780
	D-Tree	0.395	0.508	0.515	0.515	0.515
	CBB	0.425	0.640	0.741	0.790	0.825
	MCTS	**0.552**	**0.735**	**0.814**	**0.860**	**0.891**
32	Pop	0.268	0.410	0.042	0.042	0.042
	Best	0.292	0.431	0.561	0.659	0.733
	D-Tree	0.247	0.310	0.304	0.303	0.303
	CBB	0.269	0.448	0.548	0.609	0.656
	MCTS	**0.368**	**0.544**	**0.647**	**0.708**	**0.758**

Table 2.5: Mean group-estimation accuracy vs the number of items rated by a new user (iterations) for Goodreads dataset with 4,8,16 and 32 user groups.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	Pop	0.533	0.640	0.786	0.847	0.892
	Best	0.544	0.737	0.811	0.866	0.901
	D-Tree	0.612	0.615	0.615	0.615	0.615
	CBB	0.695	0.819	0.868	0.906	0.930
	MCTS	**0.717**	**0.842**	**0.900**	**0.932**	**0.951**
8	Pop	0.369	0.464	0.637	0.713	0.768
	Best	0.349	0.542	0.645	0.715	0.773
	D-Tree	0.415	0.463	0.463	0.463	0.463
	CBB	0.484	0.670	0.750	0.801	0.837
	MCTS	**0.542**	**0.701**	**0.790**	**0.847**	**0.883**
16	Pop	0.248	0.361	0.532	0.620	0.692
	Best	0.256	0.413	0.542	0.627	0.695
	D-Tree	0.298	0.345	0.344	0.344	0.344
	CBB	0.338	0.528	0.637	0.711	0.756
	MCTS	**0.423**	**0.600**	**0.709**	**0.779**	**0.828**
32	Pop	0.170	0.288	0.438	0.533	0.612
	Best	0.166	0.318	0.448	0.533	0.614
	D-Tree	0.206	0.244	0.248	0.249	0.249
	CBB	0.228	0.410	0.530	0.615	0.673
	MCTS	**0.305**	**0.494**	**0.613**	**0.701**	**0.761**

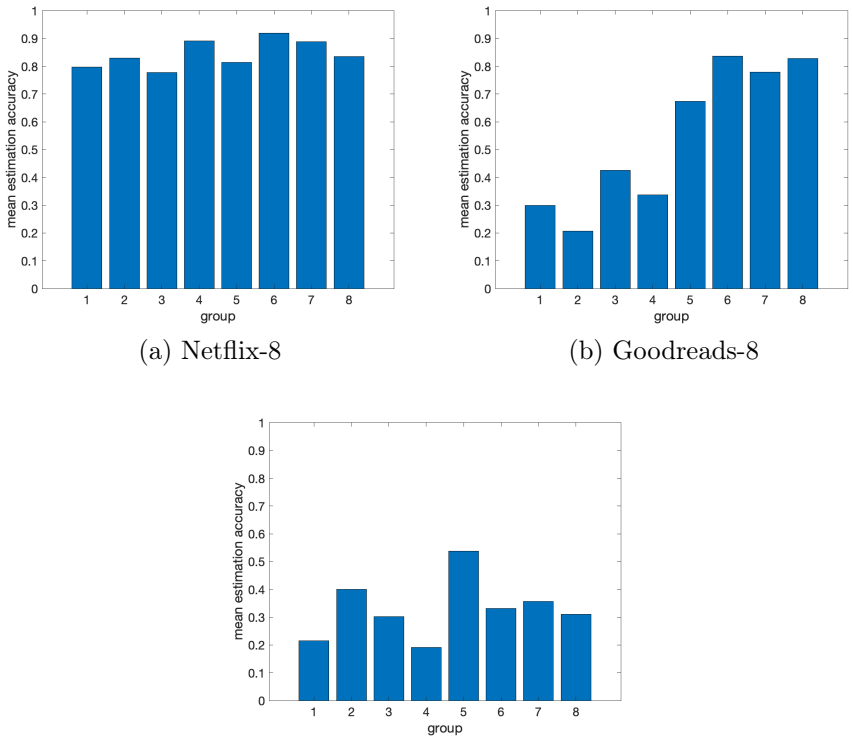
Table 2.6: Mean group-estimation accuracy vs the number of items rated by a new user (iterations) for Jester dataset with 4,8,16 and 32 user groups.

#groups	*algorithm*	*#iterations*				
		5	*10*	*15*	*20*	*25*
4	Pop	0.546	0.691	0.764	0.793	0.830
	Best	0.586	0.692	0.748	0.785	0.820
	D-Tree	0.845	0.843	0.843	0.843	0.843
	CBB	0.888	0.955	0.964	0.971	0.977
	MCTS	**0.954**	**0.987**	**0.993**	**0.996**	**0.997**
8	Pop	0.401	0.523	0.634	0.700	0.753
	Best	0.346	0.471	0.569	0.625	0.682
	D-Tree	0.595	0.680	0.678	0.678	0.678
	CBB	0.681	0.850	0.896	0.920	0.939
	MCTS	**0.815**	**0.920**	**0.951**	**0.966**	**0.974**
16	Pop	0.271	0.398	0.504	0.598	0.660
	Best	0.245	0.367	0.455	0.566	0.629
	D-Tree	0.337	0.468	0.490	0.491	0.491
	CBB	0.421	0.646	0.747	0.790	0.829
	MCTS	**0.594**	**0.772**	**0.843**	**0.879**	**0.901**
32	Pop	0.195	0.312	0.414	0.530	0.615
	Best	0.197	0.315	0.471	0.582	0.648
	D-Tree	0.234	0.290	0.299	0.299	0.299
	CBB	0.234	0.446	0.576	0.647	0.703
	MCTS	**0.378**	**0.572**	**0.659**	**0.734**	**0.775**

Table 2.7: Mean group-estimation accuracy vs the number of items rated by a new user (iterations) for Movielens10M dataset with 4,8,16 and 32 user groups.

2.4.3 Results With Ranked Comparison Feedback

Table 2.8 shows the mean group estimation accuracy measured from pairwise comparison feedback. Namely, during cold start the user is presented with a sequence of pairs of items and selects the one from each pair that they like best. The MCTS cold start algorithm selects the pairs of items to display to the user, adapting them on the fly based on the user feedback so far. Comparing with Tables 2.4-2.7 it can be seen that for a given number of iterations, the estimation accuracy is substantially reduced for all datasets. For example, after 5 iterations the Netflix accuracy is 0.733 for 4 groups and 0.579 for 8 groups compared with 0.951 and 0.838 respectively with explicit rating feedback.



(a) Netflix-8

(b) Goodreads-8

(c) Jester-8

Figure 2.7: Measured per-group accuracy for MCTS with pairwise ranked comparison feedback, on Netflix, Goodreads and Jester datasets with $t_{max} = 15$ item recommendations.

Figure 2.7 shows the estimation accuracy broken down by the group for the Netflix, Goodreads and Jester datasets. It can be seen that while the accuracy is similar across all groups for the Netflix data, there is much more variability for the Goodreads data with the accuracy ranging from approximately 0.2 to 0.85.

These results suggest that pairwise feedback is less informative than explicit rating feedback and this results in slower cold-start learning. This is, however, not the full story since in practice pairwise user feedback may be less noisy than rating

dataset	*#groups*	*#iterations*				
		5	*10*	*15*	*20*	*25*
Netflix	4	0.733	0.825	0.882	0.917	0.936
	8	0.579	0.760	0.844	0.898	0.928
Goodreads	4	0.717	0.806	0.843	0.878	0.902
	8	0.403	0.493	0.549	0.599	0.630
Jester	4	0.399	0.442	0.490	0.523	0.545
	8	0.441	0.559	0.331	0.363	0.395
Movielens10M	4	0.758	0.830	0.865	0.898	0.913
	8	0.564	0.655	0.726	0.760	0.783

Table 2.8: Mean estimation accuracy vs number of items selected by a new user, for MCTS with pairwise ranked comparison feedback, on Netflix, Goodreads, Jester and Movielens10M datasets with 4 and 8 user groups.

dataset	*#groups*	*CBB*	*MCTS*	
		avg time	*max_ turns*	*avg time*
Netflix	8	4.42ms	193312	30.90ms
Netflix	16	5.45ms	349184	86.10ms
Goodreads	8	4.43ms	151896	23.80ms
Goodreads	16	5.73ms	270528	46.78ms
Jester	8	3.75ms	79376	15.14ms
Jester	16	5.41ms	166400	19.17ms

Table 2.9: Average computation time taken to recommend next item to a new user, for Cluster Based Bandits (CBB) and MCTS with explicit item ratings.

feedback i.e. comparisons may be more reliable than absolute numerical ratings. However, we leave further evaluation of this to future work since standard datasets are not suitable and a specialised user study would be required.

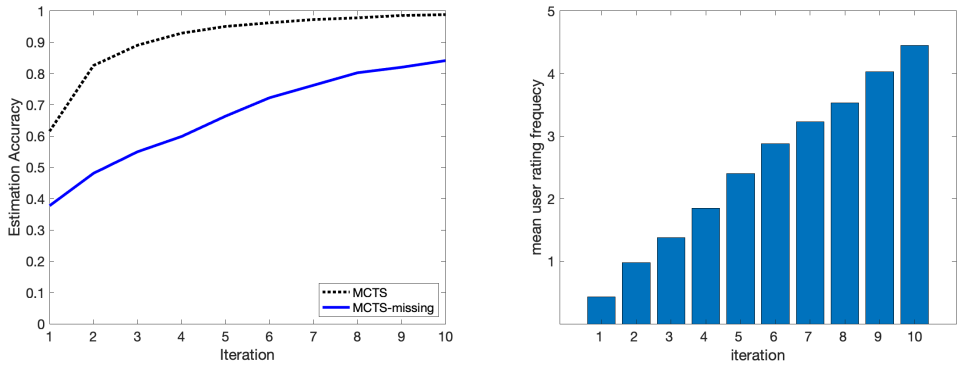
We explored asking the user to select from more than two items at each step, in the hope that this would be more informative and accelerate learning. However, since each node in the lookahead tree is a list, there is combinatorial growth in the tree branching factor as the list size is increased. As a result, the MCTS running time quickly increases. For example, Table 2.10 shows the measured running time as the number of distinguisher items $|\hat{\mathcal{V}}|$ is increased.

#groups	**#items** ($|\hat{\mathcal{V}}|$)	**#nodes** ($^{|\hat{\mathcal{V}}|}C_2$)	**average time**
4	49	1176	28 ms
8	78	3003	146 ms
16	189	17766	6.426 sec

Table 2.10: Average time taken to recommend an item using MCTS with pairwise ranked comparison feedback, for increasing number of best distinguisher items $|\hat{\mathcal{V}}|$ and number of tree nodes $^{|\hat{\mathcal{V}}|}C_2$, for Netflix dataset with 4,8 and 16 user groups.

2.4.4 Results With Missing Data

When users can choose not to rate an item, we can apply the MCTS approach with explicit ratings and simply skip learning in iterations where the user provides no feedback (and the items which are not rated are not presented again). As might be expected, this slows cold start learning. For example, Figure 2.8(a) presents measurements of the estimation accuracy when the user provides a rating at every step and when there are missing ratings. Figure 2.8(b) shows the corresponding mean number of user ratings supplied vs the number of iterations. Table 2.11 shows how the performance degrades as the number of user ratings (which is indirectly controlled by parameter α) falls.



(a) Mean estimation accuracy (b) Mean number of user ratings

Figure 2.8: (a)Baseline MCTS performance with and without missing data and (2) mean number of ratings provided at each iteration up to $t = 10$ recommendations, for Netflix dataset with 4 groups and rating frequency parameter $\alpha = 0.2$

α	*mean #ratings per user*	*mean accuracy*
0.1	2.26	0.645
0.2	4.22	0.875
0.5	7.76	0.968
1.0	10.00	0.988

Table 2.11: Baseline MCTS performance with missing ratings, mean number of ratings provided by the new users, for varying rating frequency parameter α, on Netflix dataset with 4 groups for $t_{max} = 10$ iterations

However, the fact that a user chooses not to rate an item may itself be informative of user preferences and by exploiting this extra information we can hope to gain improved cold start performance over the baseline MCTS approach that simply skips steps where the user does not rate an item. Tables 2.12-2.13 show the estimation accuracy of the baseline MCTS described above and an MCTS variant that extracts information from missing ratings using equation (2.3) (labelled "MCTS-mnar"). The

#groups	α	*algorithm*	*#iterations*				
			1	*2*	*3*	*4*	*5*
4	0.1	MCTS	0.465	0.645	0.783	0.880	0.903
		MCTS-mnar	**0.885**	**0.933**	**0.965**	**0.975**	**0.975**
	0.2	MCTS	0.680	0.875	0.935	0.968	0.983
		MCTS-mnar	**0.943**	**0.978**	**0.993**	**0.995**	**0.993**
	0.5	MCTS	0.910	0.968	0.983	0.998	0.998
		MCTS-mnar	**0.975**	**0.998**	**1.000**	**1.000**	**1.000**
8	0.1	MCTS	0.329	0.545	0.675	0.746	0.809
		MCTS-mnar	**0.771**	**0.901**	**0.938**	**0.953**	**0.963**
	0.2	MCTS	0.510	0.739	0.856	0.913	0.935
		MCTS-mnar	**0.836**	**0.941**	**0.973**	**0.983**	**0.993**
	0.5	MCTS	0.744	0.924	0.967	0.983	0.989
		MCTS-mnar	**0.868**	**0.976**	**0.992**	**0.997**	**0.999**
16	0.1	MCTS	0.352	0.573	0.726	0.792	0.846
		MCTS-mnar	**0.674**	**0.863**	**0.925**	**0.947**	**0.961**
	0.2	MCTS	0.458	0.702	0.828	0.893	0.926
		MCTS-mnar	**0.717**	**0.883**	**0.944**	**0.960**	**0.976**
	0.5	MCTS	0.609	0.833	0.905	0.946	0.960
		MCTS-mnar	**0.730**	**0.891**	**0.951**	**0.971**	**0.974**

Table 2.12: Mean group-estimation accuracy vs the number of items presented to a new user, for baseline MCTS and MCTS-mnar with varying rating frequency parameter α. Netflix movie dataset with 4, 8, and 16 user groups

#groups	α	*algorithm*	*#iterations*				
			1	*2*	*3*	*4*	*5*
4	0.1	MCTS	0.549	0.691	0.773	0.831	0.878
		MCTS-mnar	**0.714**	**0.799**	**0.839**	**0.884**	**0.894**
	0.2	MCTS	0.735	0.843	0.896	0.941	0.965
		MCTS-mnar	**0.888**	**0.938**	**0.968**	**0.979**	**0.981**
	0.5	MCTS	0.879	0.935	0.969	0.978	0.989
		MCTS-mnar	**0.905**	**0.961**	**0.979**	**0.985**	**0.995**
8	0.1	MCTS	0.548	0.696	0.786	0.841	0.875
		MCTS-mnar	**0.713**	**0.840**	**0.891**	**0.921**	**0.939**
	0.2	MCTS	0.580	0.770	0.840	0.880	0.905
		MCTS-mnar	**0.700**	**0.818**	**0.869**	**0.906**	**0.930**
	0.5	MCTS	0.679	0.826	0.884	0.904	0.913
		MCTS-mnar	**0.718**	**0.829**	**0.893**	**0.918**	**0.943**
16	0.1	MCTS	0.298	0.486	0.634	0.723	0.764
		MCTS-mnar	**0.576**	**0.735**	**0.791**	**0.838**	**0.857**
	0.2	MCTS	0.403	0.626	0.733	0.803	0.833
		MCTS-mnar	**0.619**	**0.769**	**0.849**	**0.900**	**0.918**
	0.5	MCTS	0.527	0.728	0.803	0.854	0.874
		MCTS-mnar	**0.590**	**0.757**	**0.841**	**0.884**	**0.909**

Table 2.13: Mean group-estimation accuracy vs the number of items presented to a new user, for baseline MCTS and MCTS-mnar with varying rating frequency parameter α. Goodreads books dataset with 4, 8, and 16 user groups.

rating frequency of the users is controlled by parameter α (increasing α increases the rating frequency). It can be seen that the MCTS-mnar algorithm yields considerably improved performance consistently across all datasets. As might be expected, the gain is especially large when there are many missing ratings i.e. when α is small.

2.5 Conclusions

In this research chapter, we revisit the cold-start task for new users of a recommender system whereby a new user is asked to rate a few items with the aim of discovering the user's preferences. We propose using a Monte Carlo Tree Search (MCTS) based approach to dynamically select the sequence of items presented to a new user. We find that this new MCTS-based cold-start approach is able to consistently quickly identify the preferences of a user with significantly higher accuracy than with either a decision-tree or a state-of-the-art bandit-based approach without incurring higher regret i.e the learning performance is fundamentally superior to that of the state of the art. The MCTS approach is flexible in the sense that it can readily extended to incorporate different types of user feedback including explicit ratings, ranked comparisons and missing not at random data. This boost in recommender accuracy is achieved in a computationally lightweight fashion.

Chapter 3

User-Cold Start and Item Recommendation using Reinforcement Learning Transformer: RLT4Rec

In this chapter, we introduce a new sequential transformer reinforcement learning architecture RLT4Rec and demonstrate that it achieves excellent performance in a range of item recommendation tasks. RLT4Rec uses a relatively simple transformer architecture that takes as input the user's (item, rating) history and outputs the next item to present to the user. Unlike existing Reinforcement Learning (RL) approaches, there is no need to input a state observation or estimate. RLT4Rec handles new users and established users within the same consistent framework and automatically balances the "exploration" needed to discover the preferences of a new user with the "exploitation" that is more appropriate for established users. Training of RLT4Rec is robust and fast and is insensitive to the choice of training data, learning to generate "good" personalised sequences that the user tends to rate highly even when trained on "bad" data.

RLT4Rec implementation is available at https://github.com/dilina-r/RLT4Rec.

3.1 Introduction

In this chapter, we revisit the use of Reinforcement Learning in recommender systems and introduce RLT4Rec. RLT4Rec adapts the sequential transformer neural network architecture with self-attention, which has proven to be so effective in natural language tasks, and uses it to generate high-quality item recommendations in a simple and robust manner.

The intuition for using a sequential transformer approach is that transformers have already demonstrated their ability to generate "good" token sequences (for a suitable definition of "good" e.g. in large language models it might be "grammatical human-like sentences"). Recommender systems also aim to generate "good" token sequences in the sense that the tokens are now items and we want the items to attract high user ratings. It is therefore natural to consider applying transformer models to directly generate good item sequences.

The generation of good item sequences can also be formulated as a Reinforcement Learning (RL) task. The items correspond to the actions in RL jargon, and the information gained about a user from observations of their past (item, rating) pairs is the RL state[1]. The RL reward to be maximised is the sum of the item ratings.

However, while there has been a good deal on interest in formulating recommendations as an RL task, to date RL techniques have not been widely adopted in practice. In part this is because of the difficulty of applying RL classical methods (such as Deep Q-Learning, MCTS, Actor-critic methods) for solving the recommendation task. For example, in recommender tasks the correct formulation of the user information state is often unclear, direct state observations are lacking and although one can try to estimate the state this adds to the complexity of the RL solution. Similarly, classical RL techniques are online methods i.e. in a recommender system they learn by interacting with live users. But such online training often provides a poor user experience, making it impractical. This has spurred consideration of offline RL where training uses previously collected data, but offline RL is known to suffer from thorny bias and convergence issues (Levine et al., 2020; Sutton & Barto, 2018).

By directly tackling the item recommendation task, our sequential transformer approach bypasses the intermediate Markov Decision Process (MDP) formulation step in RL and avoids many of the difficulties experienced with the application of classical RL techniques to recommender systems. We are not the first to notice the potential for using sequential transformer methods for RL tasks generally, see for example the decision transformer (L. Chen et al., 2021) and trajectory transformer (Janner et al., 2021), but there has been surprisingly little work on their

[1]This can be augmented with context information such as the user location, demographics etc.

application to RL for recommender systems. The bulk of the recommender system literature on transformers applies these to sequence-based recommendation rather than to RL, e.g. see Bert4Rec (Sun et al., 2019), SASRec (Kang & McAuley, 2018). In this literature, the aim is essentially to mimic item sequences from the training data rather than to solve an RL task. Recently, (S. Wang et al., 2023; Xin, Pimentel, et al., 2022) have applied decision transformer ideas to RL for recommender systems. These papers focus on offline RL training, which they recast as a supervised learning problem. However, they still require an explicit state formulation and state observations (using various sequential encoders for state estimation), and in (S. Wang et al., 2023) offline training remains complex. Neither evaluate per-user recommendation performance vs the number of items rated by a user, a key aspect in recommender systems generally and in RL in particular. They also do not consider constraints e.g. that the same item should not be recommended excessively to a user (they allow the same item to be recommended repeatedly, without limit, which may skew the performance evaluation). RLT4Rec avoids all of these issues, while also making use of a simpler neural network architecture.

3.2 Related Work

Use of Transformers in RL Generally: Classical RL approaches formulate the online learning task as a Markov-decision process. The aim is to select a sequence of actions that maximises a cumulative reward in the face of a changing environment (changes may be random in nature as well as in response to previous actions taken). This can be solved using a range of techniques, e.g. Deep Q-Learning Networks (DQN). However, recently there has been much interest in alternative approaches that make use of transformer neural networks. Probably the earliest works proposing a transformer approach to RL are Trajectory Transformer (Janner et al., 2021) and Decision Transformer (DT) (L. Chen et al., 2021), which have since led to a rapidly growing literature on transformer-based RL (eg: (Putterman et al., 2021; Xu et al., 2022; Yamagata et al., 2023)). These transformer RL methods apply a sequence of (state, reward, action) triples as input to a transformer, each triple corresponding to a particular time instant. The last elements in the sequence are a (state, reward) pair and the task of the transformer is to predict the next action i.e. to complete the final (state, reward) pair to generate a predicted (state, reward, action) triple. In all of these approaches the state is assumed to be observed and so available as an input to the transformer.

Transformers in RL for Recommender Systems: In recent years there has been much interest in the use of RL in recommender systems, see (Afsar et al., 2022) for a survey. In RL the state of a user is an information state and consists of the

history of items presented to a user and their rating (or other action e.g, a click or purchase). In the recommender system RL literature this sequence is mapped to a dense vector e.g. by use of a Long Short Term Memory (LSTM) network. See (Huang et al., 2022) for a recent survey on recommender system RL state estimation. The majority of existing work has focused on classical RL methods such as DQN and there has been much less work studying the use of transformers. In (Xin, Pimentel, et al., 2022) a single (state, reward) pair is input to a transformer-like module which then generates an action i.e. the next item to present to the user. The state is estimated using a variety of LSTM's. The main focus of the paper is on offline training, namely by reformulating the usual RL online learning task as supervised training of the transformer. In (S. Wang et al., 2023) a sequence of (state, reward, item) triples is input to a transformer. The state is estimated using an LSTM. Again, the main focus is on offline training, in this case by using Deep Deterministic Policy Gradient to generate training data from a simulator (which also outputs the state), this training data then being used for supervised training of the transformer and LSTM state estimator. The authors of (Xin, Pimentel, et al., 2022) have made their code available to use, and we use it as a baseline against which to compare RLT4Rec.

User Cold Start and RL: Section 2.2 of the thesis provides a comprehensive overview of user-cold start literature. In RL recommenders there is no need for a separate cold start strategy. Instead, users have different information states and a new user is just treated as having a different state from existing users. If we view the state as embodying the preference of a user, the RL task for a new user is to quickly learn the user's state i.e. to *explore*, and once it is more confident in the state to then make personalised recommendations i.e. to *exploit*, although since there will always be some uncertainty as to the user state the RL system will still continue to explore. The literature on RL for user cold start is, however, quite limited.

3.3 RLT4Rec

3.3.1 Learning Task

We consider a recommender system that presents a user with a sequence of items $\{v_i, i = 0, 1, 2, \ldots, t\}$, observing the user rating r_i for item v_i after it has been presented. The rating might be supplied explicitly by the user, e.g. by writing a review, or inferred from e.g. the engagement time they spend viewing a video, listening to a song or reading a news article. The aim of the recommender system is to maximise the sum of the item ratings $R_t = \sum_{i=0}^{t} r_i$. This setup can be extended to incorporate a display of a set of items rather than a single item e.g. a slate or full

display, and to other choices of utility R_t e.g. to include the cost or value of an item to the recommender itself, to account for the benefit to other users of rating a new item and so on. However, we focus here on the simplest setup with the display of single items and sum-rating utility since this already captures most of the important learning aspects and is already quite challenging.

The information state about a user at time t is embodied by the sequence of items that they have viewed and rated up to time t i.e. $\{(v_0, r_0), \ldots, (v_t, r_t)\}$. We typically also have context information such as the approximate location of the user (which country or city), demographics (age, gender) etc. This context information can be readily incorporated either by sharding the RL task, similarly to contextual bandits, or by incorporating the static attributes into the observed sequence of (v_i, r_i) pairs.

Learning task:

The learning task we consider is therefore as follows: given a sequence of (item, rating) pairs $\{(v_i, r_i), i = 0, \ldots, t\}$ predict the next item v_{t+1} to display so as to maximise the sum-rating $R_t = \sum_{i=0}^{t} r_i$.

Offline learning:

We have access to training data collected previously. This consists of sequences of (item, rating) pairs but these sequences (i) need not be "good" i.e. they might not have a high sum-rating (e.g. the existing recommender may have poor performance and/or users may fall back to searching for highly rated items themselves) and (ii) may be "off policy" i.e. are selected according to a different learning strategy from the one that we will learn. We will use this data to initially train the RL system offline, since online training learning purely from live recommendations would likely lead to recommending too many low-rated items during the initial learning phase and so be too irritating for users.

Unknown user state:

We do not have access to an explicit user state. We do not even know how to characterise this state. All we have access to is the sequence of (item, rating) pairs to date for a user. Note that in most previous offline RL studies, e.g. Decision Transformer (L. Chen et al., 2021), the state is provided explicitly by the environment. In RL applied to recommenders previous work has also either assumed the availability of an explicit state or has explicitly estimated the state thereby requiring joint design and training of the estimator and reinforcement learner, e.g. see (M. Chen et al., 2019; S. Wang et al., 2023; Xin, Pimentel, et al., 2022; K. Zhao, Zou, et al., 2023; X. Zhao

et al., 2018). To the best of our knowledge, RLT4Rec is the first transformer-based RL approach that avoids the need for a state estimate.

3.3.2 RLT4Rec Architecture

We tackle the recommender learning task directly using a transformer neural network based on a modified Generative Pre-trained Transformer (GPT) architecture (Radford et al., 2018). The architecture is illustrated schematically in Figure 3.1. We refer to this as **R**einforcement **L**earning **T**ransformer for item **Rec**ommend-ation (**RLT4Rec**).

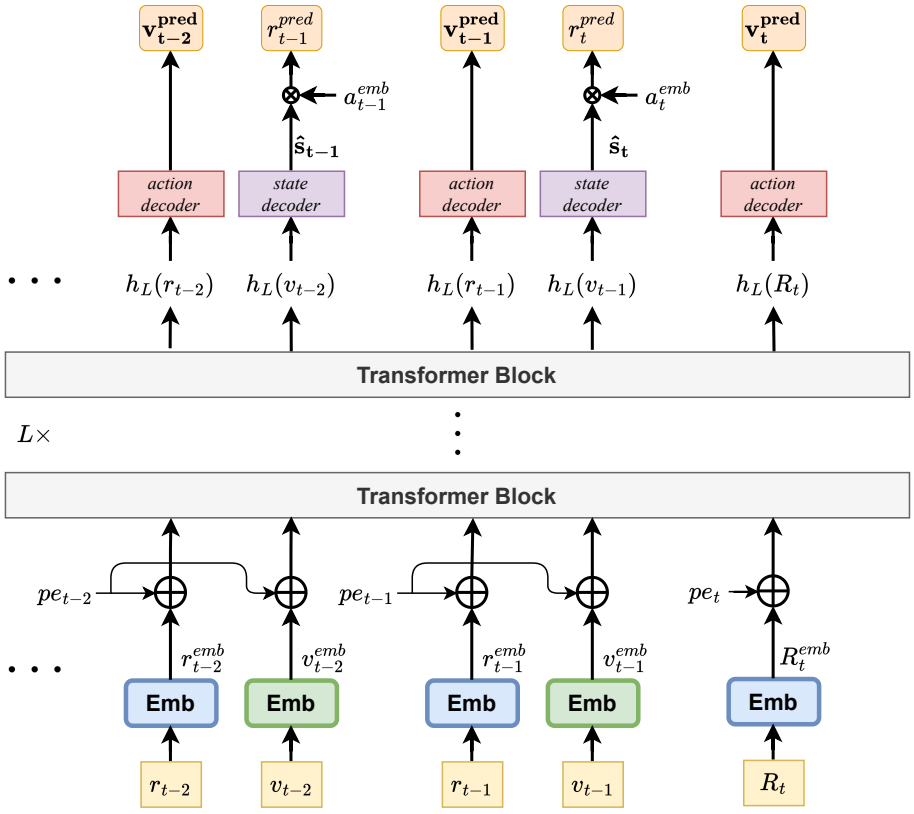


Figure 3.1: RLT4Rec architecture

This takes as input a sequence of (item, rating) pairs $\{(v_i, r_i), i = 0, \ldots, t-1\}$ plus a target rating r_t for the next item i.e. the last pair in the sequence is $(\cdot, r_t)$ and the task of the neural net is to predict the item v_t that matches the desired rating r_t. The output of the neural net is a probability vector over the set of possible items. Items predicted to have a rating closer to r_t are assigned a higher probability. The next item to recommend is then selected by choosing the item with the highest probability that has not yet been viewed by the user. Intuitively, when a large target rating is provided, the RLT4Rec will predict items that tend to yield higher ratings.

Input Embedding

We add an embedding layer at the input to map the raw (item, rating) values into dense vectors. Each item token v_i is mapped to a $\mathbb{R}^d$ vector v_i^{emb}. Similarly, each rating r_i is mapped to a $\mathbb{R}^d$ vector r_i^{emb}. A positional timestep encoding pe_i is then added to these prior to them being fed into the first transformer layer.

Multi-head Self Attention

The sequence of input tokens is fed into a stack of L transformer blocks (we use $L = 2$ blocks) with 4 attention heads each. Within each head, the input vectors are projected, using learned weights, to generate Query Q, Key K, and Value V vectors. The attention function defined in equation 3.1 maps these vectors to an output vector (Vaswani et al., 2017). We employ unidirectional transformer blocks utilizing a causal $mask$. This mask ensures the attention mechanism only attend to the previous positions, thereby avoiding any future tokens in the sequences:

$$\text{Attention}(Q, K, V, mask) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}} + mask\right)V \tag{3.1}$$

The output from the multi-head self-attention layer is passed through a feed-forward neural layer followed by $GELU$ (Gaussian Error Linear Unit) activation layer. We did not find it necessary to use skip connections.

Item Prediction

The output from the last transformer layer is passed through an action decoder module to generate one-step ahead item predictions vector $\mathbf{v_i^{pred}}, i = 0, \ldots, t$. The action decoder is a linear layer followed by a softmax activation:

$$\mathbf{v_i^{pred}} = softmax\left(\mathbf{W_v}\ h_L(r_i)\right)$$

where $h_L(r_i)$ is the transformer output corresponding to the input sequence $\{(v_0, r_0), \ldots, (\cdot, r_i)\}$, $\mathbf{v_i^{pred}}$ is the causal prediction for v_i and is a probability vector over the set of possible items and W_v is a weighting matrix.

To obtain good performance we found it necessary to modify the standard GPT architecture as follows.

Rating Inner-Product Bottleneck

During training (see below), the output from the last transformer layer is passed through an inner-product (i.e. matrix factorization) bottleneck to generate one-step ahead rating predictions $r_i^{pred}, i = 1, \ldots, t$. The inner-product bottleneck consists of

a linear layer with a tanh activation followed by an inner-product layer:

$$\hat{\mathbf{s}}_\mathbf{i} = tanh\left(\mathbf{W_s}\ h_L(v_{i-1})\right)$$
$$r_i^{pred} = \hat{\mathbf{s}}_\mathbf{i} \cdot v_i^{emb}$$

where $h_L(v_{i-1})$ is the transformer output corresponding to the input sequence $\{(v_0, r_0), \ldots, (v_{i-1}, r_{i-1})\}$, W_s is a weighting matrix, v_i^{emb} is the embedding vector for the i'th input item and r_i^{pred} is the causal prediction for the rating r_i of the i'th item. Following the usual interpretation of matrix factorization we can think of $\hat{\mathbf{s}}_\mathbf{i}$ as a user embedding vector.

This bottleneck acts in two ways: (i) it directs the learning of the item embedding v_i^{emb} and (ii) it forces the transformer blocks to learn a user embedding $\hat{\mathbf{s}}_\mathbf{i}$ that summarises the observed sequence of (item, rating) pairs. We found the addition of this bottleneck to be essential for obtaining good item predictions. To the best of our knowledge, RLT4Rec is the first transformer-based RL approach that incorporates an inner-product bottleneck.

Loss Function

RLT4Rec takes as input a sequence $\{(v_i, r_i), i = 0, \ldots, t-1\}$ of (item, rating) pairs plus a target rating r_t for the next item, which during training is taken to be the rating taken from the training data. The output of RLT4Rec is a one-step ahead item prediction vector $\mathbf{v_t^{pred}}$. In addition, during training, RLT4Rec also outputs a rating prediction r_t^{pred} (see inner-product bottleneck discussion above). The training loss function used is the sum of (i) the categorical entropy $\mathcal{L}_{CE}(\mathbf{v_t^{pred}}, v_t)$ between the item prediction $\mathbf{v_t^{pred}}$ and the one-hot encoding vector a_t for item v_t, and (ii) the square error ($\mathcal{L}_{sqr} = r_t^{pred} - r_t)^2$ between the rating prediction r_t^{pred} and the observed rating r_t for the item. This loss is then averaged over the (item, rating) sequences in the training dataset.

Training

RLT4Rec is trained using Adam with learning rate 1e-3, weight decay 1e-5 and gradient clipping (parameter=1.0). For the Netflix, Goodreads, and Movielens10M datasets a mini-batch size of 64 was used, while a mini-batch size of 32 was used for the PD1 and PD2 datasets. RLT4Rec was trained for up to 50 epochs and the best validation model was selected for evaluation. We observed that the inner-product bottleneck results in faster convergence as well as better prediction performance. All our training and performance evaluations were conducted on a machine with a NVIDIA GeForce RTX 4090 GPU and AMD Ryzen 5975WX with 32 CPU cores.

An embedding length of $d = 256$ is used for Netflix, Goodreads, Movielens10M and of $d = 128$ for the PD1 and PD2 datasets. We found $L = 2$ transformer blocks with 4 attention heads each to be sufficient to achieve good performance.

3.4 Performance Evaluation

3.4.1 Metrics

Mean rating across iterations $\overline{R}_{@t}$

The mean item rating is $\overline{R}_{@t} = \frac{1}{t}\sum_{i=0}^{t-1} r_i$ where r_i is the rating of the i'th item presented to a user. This corresponds to the sum-rating utility that we would like to maximize in the RL setup, and so is our primary performance measure.

Precision @K

The precision is the number of relevant items that are within the top-K recommendations. We define an item v to be relevant when the user-rating r of the item is higher than a specified rating threshold, which in our experiments is chosen to be 4.0.

3.4.2 Simulation Environment

To allow us to evaluate recommendation performance in a clean, reproducible manner we use a simulation environment so that we know the ground truth about user ratings.

User model

Similarly to Chapter 2, we use user-groups based simulation environments, derived from measurement datasets consisting of (user, item, ratings) triples. For each dataset, we cluster users into groups and estimate the mean $\mu(g, v)$ and variance $\sigma(g, v)^2$ of the ratings by each group g for item v. We use the BLC matrix-factorization clustering algorithm (Checco et al., 2017) for this, although other clustering algorithms (such as k-means) might also be used. To model a user belonging to group g, for each item v we draw a random gaussian rating with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. Note that the user rating for a given item is taken to be fixed since repeated ratings of the same item tend to be highly correlated. In this way, we have a full set of item ratings for the user and can generate as many different users as we like.

Public datasets

We use three public datasets, namely the Netflix dataset (480,189 people 17,770 movies, 104M ratings from 1–5), Movielens10M dataset[2] (72K people 10K movies, 10M ratings from 1–5) and the Goodreads10K dataset[3] (53,424 people rating 10,000 books, 5.9M ratings from 1-5).

Synthetic datasets

In addition to the above datasets we also use two "toy" synthetic datasets that exhibit extreme behaviour and so allow us to evaluate performance more thoroughly. In the Polarised-1 (PD1) dataset, each item tends to be given a high rating by users in just one group, with the users in other groups giving a low rating, e.g. see Table 3.1(a). The standard deviation of all ratings is $\sigma_g(v) = 0.25$. To achieve good performance (i.e. a high sum-rating) with this dataset a recommender must quickly learn the group that a user belongs to. In the Polarised-2 (PD2) dataset each item tends to be given a high rating by users in multiple groups, e.g. see Table 3.1(b). The standard deviation of all ratings $\sigma_g(v) = 0.5$. A recommender system can achieve good performance with this dataset without precisely learning the group that a user belongs to.

Training data

We create offline training data by generating a population of users (distributed across the different user groups) and then using the simulation environment to generate sequences of (item, user) ratings for these users. The set of items presented to a user is selected uniformly at random. That is, we do not train RLT4Rec on "good" (item, user) sequences yet as we will see below it nevertheless learns to generate good sequences i.e. sequences with high item ratings.

3.4.3 Baselines

We evaluate RLT4Rec against four baselines:

1. *Best**: This uses a genie that knows the group the user belongs to and so recommends a sequence of items with the highest mean ratings for that group, in decreasing order. This is an upper bound on performance, that is not achievable in practice.

[2]https://grouplens.org/datasets/movielens/
[3]https://mengtingwan.github.io/data/goodreads

groups	v_0	...	v_{25}	...	v_{50}	...	v_{75}	...
g1	5		1		1		1	
g2	1	...	5	...	1	...	1	...
g3	1		1		5		1	
g4	1		1		1		1	

(a) Polarised-1 (PD1) dataset

group	v_0	...	v_{25}	...	v_{50}	...	v_{75}	...	v_{100}	...
g1	5		1		2		3		4	
g2	4		5		1		2		3	
g3	3	...	4	...	5	...	1	...	2	...
g4	2		3		4		5		1	
g5	1		2		3		4		5	

(b) Polarised-2 (PD2) dataset

Table 3.1: Example mean ratings for items in the polarised datasets (a) PD1 and (b) PD2.

2. *Monte Carlo Tree Search (MCTS)*: We use a variant of the explicit rating based MCTS introduced in Chapter 2. This is a strong baseline that makes use of the user group structure to construct an explicit representation of the information state as a probability vector over the user groups. MCTS has the advantage of extra knowledge of the RL task (the number of user groups and the mean and variance of the item ratings in each group) that is not available to RLT4Rec, and uses this to directly search for solutions to the RL task. When run for a sufficient number of iterations it is guaranteed to find the optimal solution. See Chapter 2 for further details.

3. *Random-Uniform (R-U)*: This presents items selected uniformly at random from the items not yet viewed by the user.

4. *Random-Popular (R-P)*: This selects items with probability proportional to the number of times they have been rated by users. Popular items tend to have higher ratings.

5. *Decision Tree (D-Tree)*: A Classification and Regression Trees (CART) optimised decision Tree for user-cold start learning.

6. *Prompt-based Reinforcement Learning (Xin, Pimentel, et al., 2022) (PRL)*: An offline-RL approach for next-item recommendation. In our experiments, we use the variant of PRL with a Gated Recurrent Unit (GRU) network state encoder.

7. *Decision Transformer (DT)*: This is a vanilla Decision Transformer (L. Chen

et al., 2021), employed with a GRU network to estimate the state at each timestep.

3.4.4 Impact of varying the target item rating

RLT4Rec takes as input a sequence $\{(v_i, r_i), i = 0, \ldots, t-1\}$ of (item, rating) pairs plus a target rating r_t for the next item. It outputs a one-step ahead prediction of the next item v_t with rating r_t. As a sanity check on performance, Table 3.2 shows measurements of the mean rating of the predicted item v_t as the target rating r_t is varied. Data is shown for the Netflix and PD2 datasets but similar results are also observed for the other datasets.

For these two datasets ratings lie in the range 1 to 5. It can be seen that the mean rating of v_t quite closely tracks the target rating r_t over this range, except for the Netflix dataset when the target rating is 1 in which case the mean rating of v_t is about 2. This is because the Netflix dataset contains very few items with a rating of 1 (the ratings tend to be bunched between 3 and 5).

In summary, the mean rating of the one-step ahead item prediction by RLT4Rec closely matches the target rating r_t input to RLT4Rec. For the rest of this chapter, we set the target rating to be the maximum rating for each dataset, e.g. $r_t = 5$ for Netflix.

dataset	*target rating*	*iterations*				
		5	*10*	*15*	*20*	*25*
Netflix (8 groups)	1	1.90	1.98	2.04	2.07	2.09
	2	2.32	2.40	2.46	2.49	2.49
	3	3.16	3.18	3.21	3.24	3.24
	4	4.07	4.14	4.17	4.19	4.19
	5 (*max*)	4.51	4.47	4.42	4.38	4.37
PD2	1	1.26	1.22	1.24	1.19	1.24
	2	2.02	2.01	2.05	2.07	2.06
	3	3.00	2.96	2.94	2.90	3.06
	4	3.91	3.93	3.85	3.80	3.90
	5 (*max*)	4.60	4.63	4.63	4.62	4.60

Table 3.2: Mean rating across t ($\overline{R}_{@t}$) with varying expected returns r_t for Netflix (8 groups) and PD2(Polarised-2) Dataset

3.4.5 Recommending for cold-start users

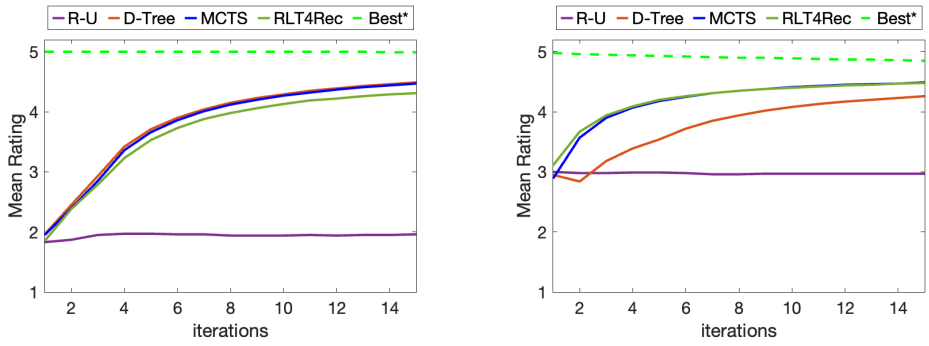
When a new user joins the system it initially has no knowledge of the preferences of the user and so would like to quickly learn these. Traditional approaches to the new

user cold-start task often employ a distinct "exploration" phase, before switching to a separate "normal" operation to make subsequent recommendations.

RLT4Rec treats cold start and normal operation in an integrated manner. In all cases the input to RLT4Rec is a sequence of user (item, rating) pairs that it uses to recommend the next item. For a new user, the sequence is empty and then grows over time as the user rates more items. *There is no need for a separate cold start mode with RLT4Rec.*

Since RLT4Rec seeks to make good item predictions for the user, when the input (item, rating) sequence is short RLT4Rec automatically tends to "explore" by choosing items that quickly reveal the user's preferences and allow RLT4Rec to learn them. This can be seen in Table 3.3, Table 3.4 and Figure 3.2, which shows the mean rating of the recommended item versus the number of items the user has rated (averaged over 200 users per group - the results are similar when we also use more users). Also shown in 3.3, Table 3.4 and Figure 3.2 are the results for Decision Tree, and MCTS. For the synthetic datasets, it can seen that the performance of RLT4Rec closely matches that of MCTS. In overall, RLT4Rec outperforms all baselines, including the MCTS, yielding higher rewards on new users even on pure-start conditions.

This good cold start performance is achieved despite the fact that MCTS has the advantage of being provided with knowledge of the user groups, including the item rating means and variances for each user group, whereas RLT4Rec lacks any information about the user groups and must instead learn it from the training data.



(a) PD1 (b) PD2

Figure 3.2: Mean rating across t ($\overline{R}_{@t}$) vs number of items t - Polarised Datasets PD1 and PD2, upto $t = 15$ item recommendations

Looking at the data in Figure 3.2 in more detail, recall that in the PD1 dataset, each item is rated highly by just one user group, and given low ratings by users in the other groups. To achieve good performance it is therefore essential for the recommender to quickly and accurately learn which group a user belongs to. It can be seen that Decision Tree and MCTS learn to recommend highly rated items after

about 5 items (after that the mean rating is always close to the maximum rating of 5). A similar behaviour can be seen with the RLT4Rec. In contrast, picking random items never achieves a mean rating of more than about 2. In the PD2 dataset each item is given a high rating by multiple user groups, and so good performance can be achieved without accurately learning the group that a user belongs to. Reflecting this, it can be seen in Figure 3.2 that random items now achieve a mean rating of 3 and both RLT4Rec and MCTS learn to recommend highly rated items after only about 2 items.

dataset	*algo*	*iterations*				
		5	*10*	*15*	*20*	*25*
Netflix (8 groups)	Best*	5.00	5.00	5.00	5.00	4.99
	R-U	3.19	3.20	3.21	3.21	3.21
	R-P	4.16	4.15	4.14	4.13	4.12
	D-Tree	3.33	3.59	3.87	4.00	4.07
	MCTS	3.84	3.97	4.02	4.05	4.07
	RLT4Rec	**4.51**	**4.47**	**4.42**	**4.38**	**4.37**
Goodreads8 (8 groups)	Best*	5.00	4.99	4.97	4.95	4.94
	R-U	3.58	3.57	3.56	3.55	3.55
	R-P	3.63	3.63	3.63	3.63	3.63
	D-Tree	3.44	3.68	3.84	3.91	3.94
	MCTS	3.84	3.88	3.87	3.86	3.85
	RLT4Rec	**4.21**	**4.20**	**4.17**	**4.14**	**4.12**
Movielens (8 groups)	Best*	5.00	5.00	5.00	4.99	4.98
	R-U	3.16	3.16	3.15	3.16	3.16
	R-P	3.60	3.59	3.59	3.59	3.58
	D-Tree	3.04	3.39	3.66	3.78	3.85
	MCTS	3.77	3.83	3.86	3.87	3.89
	RLT4Rec	**4.14**	**4.12**	**4.11**	**4.10**	**4.09**

Table 3.3: Mean ratings across t recommendations ($\overline{R}_{@t}$) with Best*, R-U(Random-Uniform), R-P(Random-Popular), D-Tree, MCTS and RLT4Rec for Netflix, Goodreads, Movielens datasets with 8 groups upto $t = 25$ item recommendations

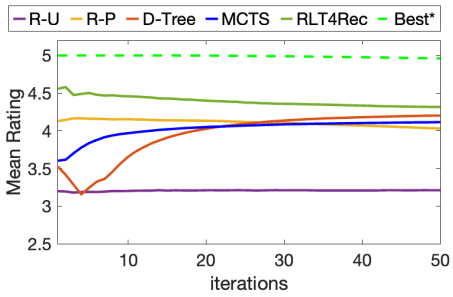
3.4.6 Performance for established users

Table 3.5 and Figure 3.3 show measurements of the longer-term performance of RLT4Rec for the Netflix, Goodreads, Movielens datasets (averaged over 200 users per group). In Table 3.5 the recommender that achieves the highest mean item rating is highlighted in bold (excluding the *Best** upper bound, which is unachievable). It can be seen from Table 3.3 and Table 3.4 that the performance of RLT4Rec consistently dominates that of MCTS and Decision Tree. This behaviour is also evident in Figure 3.3, which extends the data out to 50 items and also compares the performance against the R-P and R-U strategies.

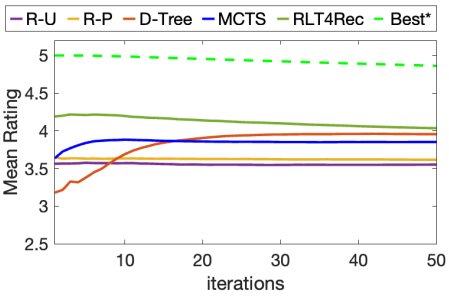
We also present measurements of the list recommendation performance of RLT4Rec

dataset	*algo*	*iterations*				
		5	*10*	*15*	*20*	*25*
Netflix (16 groups)	Best*	5.00	5.00	5.00	5.00	5.00
	R-U	3.29	3.30	3.30	3.30	3.30
	R-P	3.73	3.71	3.72	3.72	3.71
	D-Tree	3.21	3.49	3.80	3.94	4.02
	MCTS	3.84	3.93	3.97	4.00	4.02
	RLT4Rec	**4.33**	**4.35**	**4.34**	**4.32**	**4.30**
Goodreads (16 groups)	Best*	4.98	4.95	4.92	4.88	4.85
	R-U	3.58	3.58	3.58	3.58	3.58
	R-P	3.72	3.72	3.72	3.72	3.71
	D-Tree	3.52	3.66	3.81	3.87	3.89
	MCTS	3.82	3.86	3.88	3.89	3.89
	RLT4Rec	**4.05**	**4.07**	**4.05**	**4.02**	**3.99**
Movielens10M (16 groups)	Best*	5.00	5.00	4.99	4.98	4.97
	R-U	3.36	3.36	3.36	3.36	3.36
	R-P	3.53	3.53	3.53	3.53	3.53
	D-Tree	3.65	3.57	3.67	3.79	3.86
	MCTS	3.85	3.89	3.92	3.94	3.95
	RLT4Rec	**4.12**	**4.10**	**4.07**	**4.05**	**4.01**

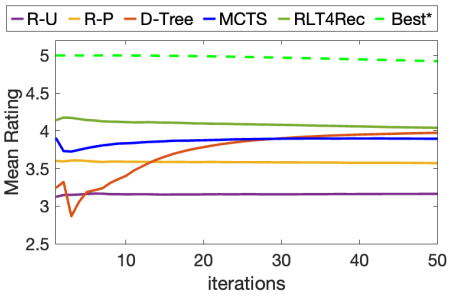
Table 3.4: Mean ratings across t recommendations ($\overline{R}_{@t}$) with Best*, R-U(Random-Uniform), R-P(Random-Popular), D-Tree, MCTS and RLT4Rec for Netflix, Goodreads, Movielens datasets with 16 groups upto $t = 25$ item recommendations



(a) Netflix-8groups



(b) Goodreads-8groups



(c) Movielens10M-8groups

Figure 3.3: Mean rating across t ($\overline{R}_{@t}$) up to 50 items, for Netflix, Goodreads, Movilens10M with 8 groups

after a user has rated 25 or more items (we take this as a rough threshold whereby a user is no longer "new"). A list of recommended items is generated by predicting a relevance score for all the items not yet viewed by the user and recommending the top K items. Recall that RLT4Rec outputs a probability vector over the set of possible items, assigning a higher probability to items that are predicted to be given a higher rating by the user. We use this probability as the relevance score for RLT4Rec. For the MCTS, we use the estimated mean reward, computed for each item during the tree search, as the relevance score. We also compare the performance against two offline-RL methods PRL (Xin, Pimentel, et al., 2022) and a vanilla Decision Transformer (L. Chen et al., 2021).

Table 3.5 shows the measured Precision@K performance as the size K of the list of recommended items is varied between 1 and 20 (K=1 represents the case when only a single item is recommended). It can be seen that again RLT4Rec consistently dominates that of PRL, Decision Transformer, MCTS, R-U and R-P.

The Decision Transformer is provided with the same information as our RLT4Rec in the input sequence, along with an additional 'state' which is generated through a GRU network. It can be seen that the decision transformer outperforms the PRL method, which also uses a GRU for state estimation and predicts an item for a given expected reward. However, RLT4Rec shows a substantial improvement over all methods, despite not being given a state estimate in the input sequence. That is, RLT4Rec is both simpler and more effective.

It is worth noting that the performance gain of RLT4Rec over MCTS is achieved despite the fact that MCTS has the advantage over RLT4Rec in knowledge of the RL task (the number of user groups and the mean and variance of the item ratings in each group) that is not available to RLT4Rec, and uses this to directly search for solutions to the RL task. When run for a sufficient number of iterations it is guaranteed to find the optimal solution. However, the computational cost of MCTS increases exponentially with the size of the set of possible items to recommend (since the lookahead search tree grows exponentially) and acts to constrain the performance achievable in practice.

3.5 RLT4Rec Learns An Information State Representation

In contrast to conventional RL systems (including previous approaches using transformers), the RL state is not provided as an input to RLT4Rec. All that is provided as input to RLT4Rec is a sequence of user (item,rating) pairs. Nevertheless, RLT4Rec succeeds at accurately predicting highly rated items, even for polarised

dataset	*algo*	*prec@1*	*prec@5*	*prec@10*	*prec@15*	*prec@20*
Netflix (8 groups)	R-U	0.346	0.342	0.340	0.340	0.341
	R-P	0.663	0.675	0.675	0.667	0.661
	MCTS	0.653	0.660	0.642	0.640	0.639
	PRL-GRU	0.683	0.763	0.759	0.753	0.751
	DT	0.826	0.800	0.776	0.758	0.750
	RLT4Rec	**0.855**	**0.830**	**0.794**	**0.770**	**0.758**
Goodreads (8 groups)	R-U	0.393	0.412	0.414	0.413	0.414
	R-P	0.423	0.442	0.441	0.438	0.439
	MCTS	0.561	0.551	0.537	0.527	0.521
	PRL-GRU	0.696	0.641	0.624	0.616	0.612
	DT	0.699	0.659	0.639	0.628	0.617
	RLT4Rec	**0.778**	**0.718**	**0.696**	**0.675**	**0.660**
Movielens10M (8 groups)	R-U	0.267	0.267	0.270	0.270	0.268
	R-P	0.393	0.392	0.392	0.392	0.391
	MCTS	0.581	0.531	0.525	0.517	0.516
	PRL-GRU	0.618	0.598	0.585	0.572	0.562
	DT	0.648	0.627	0.612	0.602	0.594
	RLT4Rec	**0.693**	**0.653**	**0.632**	**0.622**	**0.615**

Table 3.5: Precision@K for list of top-K recommendations for R-U(Random-Uniform), R-P(Random-Popular), MCTS, PRL-GRU, DT(Decision Transformer) and our RLT4Rec with established users in Netflix, Goodreads, Movielen10M datasets (8 user-groups)

datasets such as PD1 where each item is rated highly by just one user group and so for good performance it is essential that the recommender learns which group a user belongs to. It therefore seems that RLT4Rec learns an internal representation of the user information state that it then uses when making predictions. In this section, we investigate this further.

3.5.1 User Information State

For the RL task that we consider here, the relevant state is the user information state. In particular, in the user model in Section 3.4 each user belongs to one of a set of groups and the information state after rating a sequence of t items is the vector $P^{(t)} := (p_1^{(t)}, \ldots, p_n^{(t)})$ where $p_g^{(t)}$ is the probability that the user belongs to group g given the observed item ratings.

Recall, the rating of item v by a user in group g is a Gaussian random variable with mean $\mu(g, v)$ and variance $\sigma^2(g, v)$. For an observed sequence $D^{(t)}$ of ratings, we can calculate the ground-truth value of this state vector using the eq 2.1.

3.5.2 Reconstructing the State From RLT4Rec Activations

To investigate whether RLT4Rec learns the state $P^{(t)}$ corresponding to an (item, rating) sequence we use a *probe* (Belinkov, 2018, 2022). Namely, we train a Multi-Layer Perceptron (MLP) neural network with 2-layers to predict the state $P^{(t)}$ from the neuron activations within RLT4Rec. In particular, we use the $\hat{s}_t$ activations from the inner-product bottleneck in RLT4Rec as input to the probe MLP, since these are the obviously analogous to a user embedding vector within RLT4Rec. We create training data for the probe by generating (item, rating) sequences for random items and users in each group.

If the probe MLP is able to accurately predict the state $P^{(t)}$ from the RLT4Rec internal activations, this suggests that RLT4Rec has indeed learned a representation of the state. As a baseline for comparison we extract the hidden layer outputs from an untrained RLT4Rec model, initialised with random model parameters.

3.5.3 State Reconstruction Results

Table 3.6 shows the measured Mean Absolute Error (MAE) $\frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}\left|P_g^{(t)}-\hat{P}_g^{(t)}\right|$ between the group membership $\hat{P}^{(t)}$ predicted by the probe MLP and the true group membership probabilities $P^{(t)}$. It can be seen that after training the MAE for RLT4Rec is generally an order of magnitude smaller than before training, consistent with the hypothesis that trained RLT4Rec model is indeed representing the state within its internal $\hat{s}_t$ activations.

Table 3.7 also shows the measured group estimation accuracy i.e. the fraction of times that the user group $\hat{g} = \text{argmax}_{h\in\mathcal{G}}\, P_h^{(t)}$ estimated to have highest probability by the probe is in fact the correct user group. It can be seen that after training, the accuracy of RLT4Rec rises to be close to 100% after 5 items have been rated whereas, for the untrained RLT4Rec, the accuracy remains around 25% regardless of the number of items rated. Again, this evidence is consistent with the hypothesis that the trained RLT4Rec model has learned the user state.

dataset	*method*	*iterations*				
		1	*2*	*3*	*4*	*5*
PD1	Random	0.199	0.276	0.321	0.345	0.359
	Trained RLT4Rec	0.003	0.003	0.005	0.005	0.005
PD2	Random	0.231	0.274	0.293	0.301	0.305
	Trained RLT4Rec	0.053	0.035	0.022	0.016	0.012

Table 3.6: Mean Absolute Error for Probe $\hat{P}_G^{(t)}$ predictions for Random (untrained RLT4Rec with randomly initialized model parameters) and Trained RLT4Rec, for PD1 and PD2 Datasets

dataset	*method*	*iterations*				
		1	*2*	*3*	*4*	*5*
PD1	Probe-Random	0.250	0.262	0.256	0.252	0.254
	Probe-RLT4Rec	0.506	0.676	0.826	0.907	0.954
	True $P_g^{(t)}$	0.502	0.682	0.822	0.909	0.957
PD2	Probe-Random	0.239	0.248	0.256	0.264	0.271
	Probe-RLT4Rec	0.729	0.875	0.940	0.967	0.982
	True $P_g^{(t)}$	0.748	0.889	0.949	0.972	0.988

Table 3.7: Group-estimation accuracy, calculated from predicted Group Membership $\hat{P}_g^{(t)}$ with Random (Untrained) and RLT4Rec (trained) against the True Probabilities $P_g^{(t)}$, for PD1 and PD2 Datasets

3.6 Investigating the effect of a "good" state estimator

An accurate user representation is essential in recommender systems, as it reflects their individual preferences, interests, and behaviour. It can allow the system to deliver more relevant and personalized recommendations. In RL-based models, this is achieved through state-estimation methods and (Huang et al., 2022; Xin, Pimentel, et al., 2022) observe that state-estimation can have a substantial impact on recommender performance.

We conduct several experiments, independent from the RLT4Rec, to investigate whether an *ideal-state* can yield a much simpler recommender model.

3.6.1 MLP Value Network

In reinforcement learning, value-based methods such as Deep Q-Networks make use of deep learning models to predict the Q-values of all actions or Q-value of a given state-action pair. As a simpler alternative, we consider the Multi-Layer Perceptron (MLP) recommender shown in Figure 3.4, which acts similarly to a Value network.

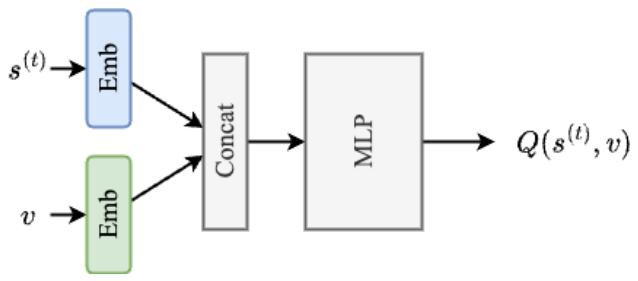


Figure 3.4: MLP Value Network Architecture

The input to the MLP is a (state, item) pair (s^t, v). The output is the predicted reward for item v. We select the item with the highest reward not yet rated by the user as the recommendation. The state s^t and item v inputs are mapped to d dimensional vectors via an embedding layer. We use an embedding size of $d = 128$. The concatenated input embeddings are passed through an MLP with one hidden layer having the number of neurons equal to the embedding size. The output is the predicted reward for the input item. As for the training data, we use the same simulation setup in Section3.4.2 to generate offline datasets for training. During training, we minimize mean square error loss between the predicted reward and the true reward (i.e: item rating) in the training sequence.

State Estimation: Incorporating trainable state-encoders, such as Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs), can be commonly seen in RL literature. These need to be trained concurrently with the RL-models. As a result of this, the state generated by the encoders will differ across models since the learned weights are specific to each training instance. In order to focus on the RL aspect of cold start recommendation, we construct our experiments so that the ground truth user state is known. We considered the group-membership $p_g^{(t)}$ (determined by eq 2.1 in Section 2.3.1) based state to be a good state estimate to conduct our experiments in a controlled and reproducible manner.

3.6.2 MLP Outperforms Other Methods

We conduct evaluations in a cold-start scenario, as the user-state tends to be highly uncertain and quickly changes with each item-interaction. For every new user, the system initially presents an item to the user and in turn the user provides a feedback (i.e: item rating). The system then continues to present the user with items, while learning from the feedback to the previously recommended items. To evaluate the cold-start performance, we generate new users belonging to each group $g \in G$, from the simulation environment.

Table 3.8 shows the mean reward for between 5 and 25 item recommendations measured for the Netflix, Goodreads and Movielens10M datasets with 8 and 16 groups. The highest values are indicated in bold. Note that this is a fair comparison in the sense that all of the approaches have access to the same user data, including the same state estimate.

Surprisingly, at least to us, it can be seen that the MLP value network consistently achieves better performance compared to PRL. The Decision Transformer and MLP are on par, each showing superior performance under different datasets and conditions. This is despite the fact that the MLP is considerably simpler than the Decision Transformer and PRL architectures.

dataset	*algo*	*iterations*				
		5	*10*	*15*	*20*	*25*
Netflix 8	R-U	3.231	3.219	3.214	3.211	3.213
	PRL	4.546	4.475	4.430	4.409	4.387
	DT	4.559	4.487	4.445	4.422	4.407
	MLP	**4.566**	**4.498**	**4.463**	**4.434**	**4.417**
Goodreads 8	R-U	3.535	3.542	3.546	3.545	3.543
	PRL	4.140	4.148	4.130	4.103	4.082
	DT	4.260	4.218	4.189	4.164	4.146
	MLP	**4.268**	**4.237**	**4.204**	**4.173**	**4.149**
Movielens 8	R-U	3.166	3.157	3.161	3.160	3.159
	PRL	4.096	4.083	4.082	4.068	4.062
	DT	4.179	4.149	4.135	4.119	**4.109**
	MLP	**4.181**	**4.152**	**4.138**	**4.122**	4.106
Netflix 16	R-U	3.319	3.325	3.322	3.321	3.321
	PRL	4.465	4.404	4.352	4.344	4.327
	DT	**4.532**	4.437	4.393	4.373	4.362
	MLP	4.531	**4.451**	**4.407**	**4.385**	**4.368**
Goodreads 16	R-U	3.604	3.604	3.606	3.609	3.609
	PRL	4.189	4.120	4.087	4.051	4.013
	DT	**4.241**	4.188	4.148	**4.121**	**4.104**
	MLP	4.240	**4.194**	**4.149**	4.118	4.091
Movielens 16	R-U	3.360	3.360	3.369	3.368	3.367
	PRL	4.153	4.089	4.046	4.027	4.017
	DT	4.191	**4.172**	**4.166**	**4.162**	**4.154**
	MLP	**4.194**	4.168	4.148	4.134	4.121

Table 3.8: Mean ratings across t recommendations ($\overline{R}_{@t}$) with Random-uniform (R-U), MCTS, PRL, Decision Transformer (DT) and Value Network (MLP) for Netflix, Goodreads, Movielens10M datasets with 8 and 16 groups

This has significant implications. In particular, it means that there is little value to be gained by adding complexity to the RL component of DT and PRL recommenders. Indeed in our experience adding extra complexity may lead to a loss in performance. For the Decion Tree this loss in performance is presumably due to the greater difficulty of training.

By design, the state is explicitly known in these tests so as to avoid confounding effects associated with state estimation. We leave evaluation of the impact of state estimation errors to future work, but note that our results suggest that the primary effort in RL recommenders should be devoted to designing good state estimators. In contrast, most of the current recommender RL literature focuses on designing the mapping from (state, reward, item) triplets to the next item, which of course is also the main focus of the RL literature generally.

It is also worth noting that the Decision Transformer not only receives a state input at each step, but also receives the (item,reward) pairs of the user's past interactions. The history of past (item,reward) pairs is enough to allow the state to be

calculated.

3.6.3 Computational Cost

In addition to achieving strong recommendation performance, the MLP value network is also cheaper computationally than the Decision Transformer and PRL. This can be seen from Table 3.9 which shows the measured compute time of the MLP, DT and PRL for the Netflix dataset as the size of the set of items available to be recommended is varied. The compute time shown is the average time to recommend 25 items to a user, including the state estimation and model prediction times. Also shown in Table 3.9 is the GPU memory used by each approach during the evaluations.

dataset	*#items*	*no. of parameters*			avg time (ms)			GPU mem (MB)			$\overline{R}_{@25}$		
		MLP	*DT*	*PRL*	*MLP*	*DT*	*PRL*	*MLP*	*DT*	*PRL*	*MLP*	*DT*	*PRL*
Netflix-S	752	34.7K	547K	227K	8.92	43.89	14.67	8.63	11.48	8.99	**4.417**	4.407	4.387
Netflix-M	4,804	34.7K	1.07M	750K	11.60	43.01	14.86	10.60	16.62	10.99	4.374	**4.380**	4.326
Netflix-L	15,020	34.7K	2.4M	2.07M	15.92	43.52	15.20	15.59	29.55	16.01	4.358	**4.361**	4.298

Table 3.9: Average recommendation time per user, model parameter count, GPU memory usage, Mean reward $\overline{R}_{@25}$ for the Value Network (MLP) and Decision Transformer (DT), with varying available number of items - Netflix 8 groups dataset

This is perhaps unsurprising given the simplicity of the MLP, but recall that the MLP outputs the predicted rating for a single item and so to predict the next item to recommend it needs to be run for every item not yet rated by the user. Thus, the computational cost of the MLP increases linearly with the number of items, although this might be compensated by implementing batch-predictions. The Decision Transformer and PRL both output a probability vector over the set of possible items, which scales with the number of items. However, it can be seen from Table 3.9 that the compute time for DT and PRL increases slightly with the number of items and so the compute burden is dominated by other components of these networks.

The last column of Table 3.9 also summarises how the recommendation performance varies with the number of available items. It can be seen that the performance tends to fall as the number of items is increased, presumably reflecting the increased difficulty of finding the best item to recommend when the pool of available items is larger. However, it can also be seen that the decrease in performance is much more pronounced for PRL compared to the MLP and DT.

3.7 Conclusion

In summary, we introduce a new sequential transformer architecture RLT4Rec and demonstrate that it achieves state of the art performance in a range of item recommendation tasks.

RLT4Rec uses a relatively simple transformer architecture that takes as input a sequence of user (item,rating) pairs and outputs the next item to present to the user. The RLT4Rec architecture includes an inner-product bottleneck, and we find that this plays a key role in achieving good performance. In particular, our analysis indicates that this bottleneck allows RLT4Rec to construct an internal state representation that successfully captures user preferences. There is no need to input an explicit state observation or estimate to RLT4Rec. This not only simplifies the recommender system architecture but also avoids the difficult design step of deciding upon an appropriate user state representation since this is now learned automatically.

We show that RLT4Rec handles new users and established users within the same consistent framework and automatically balances the "exploration" needed to discover the preferences of a new user with the "exploitation" that is more appropriate for established users. RLT4Rec consistently improves upon the performance of PRL (Xin, Pimentel, et al., 2022) for established users and MCTS for new users.

Training of RLT4Rec is robust and fast and is insensitive to the choice of training data. It can be trained using random sequences of items yet nevertheless learns to generate "good" personalised sequences that the user tends to rate highly. While offline RL learning is known to be difficult, our results show that this may be much less troublesome for RLT4Rec.

Chapter 4

Reassessing the Effectiveness of Reinforcement Learning based Recommender Systems for Sequential Recommendation

Over the past few years, researchers have explored the use of reinforcement learning (RL) for sequential recommendation problems. However, since RL techniques commonly target at optimizing long-term rewards, it is surprising that RL-based models are reported to be competitive with traditional supervised models when evaluated under the myopic next-item prediction protocol. A recent study suggests that reported performance gains of combining RL with supervised learning techniques, as done in the Self-Supervised Q-Learning (SQN) framework, may actually not come from learning an optimal policy, but that the RL component helps to learn embeddings that encode the users' past interactions. Given these observations, we aimed to reassess the performance of RL-enhanced sequential recommendations in the SQN framework. While we were able to reproduce the results reported in the respective papers, we found that properly-tuned supervised learning models like GRU4Rec substantially outperform the proposed RL-models from the literature. Our analyses furthermore revealed that there is a significant inconsistency in terms of evaluation protocols in the literature and that the use of third-party implementations of existing models may lead to unreliable conclusions. Overall, still more research and alternative evaluation schemes seem required to fully leverage the power of RL for sequential recommendation tasks.

The implementations and online resources for this chapter can be found in https://github.com/dilina-r/rl-rec.

4.1 Introduction

Over the past few years, sequential recommendation tasks have received increased attention in the literature (Quadrana et al., 2018; S. Wang et al., 2021). Correspondingly, a rich variety of technical approaches have been explored for the task, including both heuristic techniques based on nearest-neighbours as well as deep learning models based on various underlying architectures like recurrent neural networks, attention, or transformers (Hidasi et al., 2016; Kang & McAuley, 2018; J. Li et al., 2017; Ludewig et al., 2021; Sun et al., 2019). Besides such *supervised learning* techniques, researchers have also increasingly begun to explore the use of *reinforcement learning* (RL) for sequential recommendation tasks by considering the task as a sequential decision-making problem (Afsar et al., 2022; X. Chen et al., 2023). RL can be seen as a *natural* fit for this problem, as it matches the dynamic and sequential nature of the problem and is able to focus on the long-term effects of the system, e.g., on user engagement.

In terms of the evaluation of sequential recommendation models, supervised models are commonly benchmarked in offline settings under the *next-item prediction* (NIP) protocol. In this protocol, the model is provided with the last sequence of observed user interactions, e.g., from a current usage session, and the task is to predict the user's next interaction, e.g., an item-view or purchase action in an e-commerce use case. In a number of research works (e.g., (Labarca et al., 2024; C. Li et al., 2023; Ren et al., 2023; Xin, Pimentel, et al., 2022; Xin et al., 2020)), the NIP protocol has also been adopted for models that are based on reinforcement learning instead of relying on the cumulative reward as used in traditional RL evaluation setups.

However, as discussed in (Deffayet et al., 2023), there is an apparent mismatch between using a model that is designed to maximize long-term reward and at the same time relying on a myopic evaluation protocol that focuses on the correct prediction of the immediate next user action. As such, it may appear surprising that using RL for next-item prediction can actually improve the performance of supervised models in terms of short-term metrics, as reported, for example, for the Self-Supervised Q-Learning (SQN) framework proposed in (Xin et al., 2020). In a recent work published at ICML '24, (Labarca et al., 2024) therefore investigate the "*unexpected effectiveness*" of RL for sequential recommendation, using the SQN framework as an example. From their analyses, the authors conclude that the RL component of the hybrid framework that is added to a supervised model may actually not optimize some long-term reward as probably intended, but "*promotes the learning of embeddings that capture information about the user's previously interacted items.*" To provide evidence to support this hypothesis, they exchanged the RL component

with a comparably simple auxiliary loss that for example predicts the *number of previously interacted items* of the user. Their results indeed show that this alternative component can lead to similar performance gains as the RL-based loss component.

Inspired by these recent observations, we aimed to further explore this phenomenon. We started with an effort to reproduce the reported results using the artifacts that were shared by the authors in (Labarca et al., 2024) and which are based on the artifacts developed and shared by the authors of the SQN framework (Xin et al., 2020). Furthermore, since the experiments in (Xin et al., 2020) are mainly based on demonstrating the relative performance gain obtained through the RL component over a given underlying supervised model, we aimed at considering alternative baselines as reference points in our own evaluation. Specifically, we finally included both comparably simple, yet often effective methods in such an evaluation, as well as an alternative neural model (NARM) (J. Li et al., 2017), which also led to competitive results in previous evaluations (Ludewig et al., 2021). Furthermore, when examining the shared artifacts, we found that they included alternative implementations of the GRU4Rec (Hidasi et al., 2016) and SASRec (Kang & McAuley, 2018) models. Thus, we extended our set of comparison methods with the official author-provided GRU4Rec implementation.

In this extended evaluation, we could successfully reproduce the numbers reported in (Labarca et al., 2024), also confirming the performance gains obtained by adding the RL component to the underlying models. However, we found that the official GRU4Rec implementation substantially outperformed all SQN-variations from (Xin et al., 2020) and (Labarca et al., 2024). Thus, while the gains reported in (Xin et al., 2020) and (Labarca et al., 2024) can be confirmed, they were apparently only achieved over weak backbone model implementations. In fact, the performance of the alternative GRU4Rec implementation was markedly lower than the original version, a phenomenon that was recently also identified for other GRU4Rec implementations in the literature (Hidasi & Czapp, 2023). Furthermore, our analysis revealed major inconsistencies in the literature, e.g., regarding the used data splitting procedures.

Overall, the use of alternative model implementations and the lack of a standardized benchmark for evaluations make it difficult to assess the true progress that is made by adding an RL component to existing supervised models. Still, these problems should not shy us away from further exploring the use of RL for sequential recommendation problems. It is however important to ensure that future experimental comparisons rely on comparable experimental configurations and involve a representative set well-tuned baselines. Furthermore, other evaluation protocols than next-item prediction should be more frequently explored for models based on reinforcement learning, as suggested in (Deffayet et al., 2023).

This Chapter is organized as follows. Next, in Section 4.2, we summarize our efforts of reproducing the findings from (Labarca et al., 2024; Xin et al., 2020), and we report additional observations that were made in this process. In Section 4.3, we then describe the design of a series of experiments in which we benchmark the proposals from (Labarca et al., 2024; Xin et al., 2020) with additional baselines and alternative evaluation protocols. Section 4.4 presents the obtained results, which we discuss in Section 4.5.

4.2 Reproducibility of SQN Results

4.2.1 Background

The recent work of (Labarca et al., 2024) studies the effectiveness of "*self-supervised reinforcement learning for recommender systems*", an approach that was presented at SIGIR '20 (Xin et al., 2020). In this earlier paper, actually two *frameworks* are proposed: (1) **SQN** (Self supervised Q-Learning) and (2) **SAC** (Self supervised Actor-Critic). Technically, the authors introduce a reward-driven RL head to an existing supervised-learning model, noting that "*directly applying standard RL algorithms to recommender systems data is essentially unfeasible.*" Using objectives like Double Q-Learning (in SQN) and Actor-Critic (in SAC), the RL component acts as a regularizer, guiding the supervised layer to align with reward-driven objectives. One central element of both approaches are the loss functions, which are used to combine and jointly train the loss from the supervised model (L_s) and the loss of the RL component (L_q) based on the given implicit feedback data. For the SQN variant, the loss is defined as follows.

$$L_{SQN} = L_s + L_q \tag{4.1}$$

The authors add the SQN and the SAC methods to four supervised learning baselines: GRU4Rec (Hidasi et al., 2016), Caser (Tang & Wang, 2018), NItNet (F. Yuan et al., 2019) and SASRec (Kang & McAuley, 2018). The proposed methods are then evaluated on two widely used datasets for session-based recommendation, and the results suggest that the SQN and SAC methods show significant improvements over the respective supervised baselines, which are used as backbones. Several follow-up works applied similar principles for leveraging RL for recommendation tasks (Antaris & Rafailidis, 2021; Gao et al., 2022; Stamenkovic et al., 2022; Xin, Karatzoglou, et al., 2022), as mentioned in (Labarca et al., 2024).

The authors of (Labarca et al., 2024) base their analyses on the findings and shared artifacts from (Xin et al., 2020). From the two frameworks, they focus on the SQN variant. As mentioned, their hypothesis is that the observed *unexpected*

performance gains under a myopic evaluation protocol do *not* stem from the RL component's guidance towards long-term reward, and they provide a theoretical analysis that shows that an optimal policy actually "*might perform arbitrarily worse than the self-supervised model [...] according to NIP metrics.*" Instead, they hypothesize that the RL component implicitly learns to encode information about the individual user's past preferences.

Labarca et al. then explore different variations of the SQN model by alternating the way of computing L_q in Eq. 4.1. One variant, named L_{eval}, corresponds deriving Q-functions from experiences based on *Temporal Difference* learning. As an alternative, the goal could be to estimate the *optimal* Q-function, referred to in the paper as L_{DQN}. According to the authors, a key advantage of exploring L_{eval} is that this approach can be easier analyzed as L_{DQN}, and an empirical analysis shows that the performance of L_{eval} does not drop significantly compared to L_{DQN}.

Ultimately, however, the authors suspect that L_{DQN} is not learning a good approximation of the *optimal* Q-function, due to the general challenges of offline RL. They thus decided to search for alternative features that correlate well with the q-values learned through L_{DQN}. Specifically, they engineered twelve features, of which they retained five as promising candidates after a correlation analysis. Interestingly, the most promising feature in their analysis turned out to be the number of past interactions in the given sequence. They then replace the RL component of the SQN architecture with a history length prediction component (named `HIST`), and this head is trained using the mean squared error between the prediction and the target number.

Through a series of experiments on the same datasets used as in (Xin et al., 2020) the authors ultimately find that comparable performance gains over the supervised backbones can be achieved by replacing the originally proposed RL component with the described alternatives. They interpret this finding as evidence that the RL process of finding an optimal policy (Xin et al., 2020) might actually not be the reason for the observed performance improvements. Instead, as mentioned, they speculate that the RL process helps to encode information about the user's previous interacted items.

4.2.2 Reproducibility Results and Further Observations

Reproducibility The authors of (Labarca et al., 2024) and (Xin et al., 2020) provide a link to the implementation of the models and the preprocessed datasets. Furthermore, the code for the baselines and data preprocessing scripts are shared as well. An earlier reproducibility study (Cavenaghi et al., 2023) analyzes the reproducibility of the code shared for the SQN model using ACM's definition of re-

producibility[1], concluding that the code is *exercisable*. This means that the code is executable but does not output the evaluation results reported in the original paper. We made the same observation as in (Cavenaghi et al., 2023), and we thus had to apply certain modifications to the provided code to obtain results that are close to the originally reported outcomes. The deviation between our results and those reported in the paper after adapting the provided code and data were in the large majority of cases less than five percent. The obtained reproducibility results using the code and protocol from (Labarca et al., 2024) and (Xin et al., 2020) are show in Table 4.1 for *clicks*, and in Table 4.2 for *purchase* events.

	Reproduced Results				ICML'24 Paper results			
	Yoochoose (RC15)		RetailRocket		Yoochoose (RC15)		RetailRocket	
	H@20	N@20	H@20	N@20	H@20	N@20	H@20	N@20
SASRec	0.494	0.276	0.233	0.145	0.496	0.275	0.231	0.145
SQN-SAS	0.498	0.273	0.260	0.161	0.505	0.280	0.260	0.162
EVAL-SAS	0.508	0.282	0.257	0.159	0.502	0.278	0.258	0.161
GRU	0.439	0.239	0.175	0.108	0.427	0.231	0.175	0.109
SQN-GRU	0.458	0.252	0.200	0.128	0.457	0.251	0.200	0.127
EVAL-GRU	0.456	0.251	0.201	0.128	0.457	0.251	0.201	0.128

Table 4.1: Reproduced Results against results reported in ICML'24 paper (Labarca et al., 2024) for *Click*-events for Yoochoose and RetailRocket datasets

	Reproduced Results				ICML'24 Paper results			
	Yoochoose (RC15)		RetailRocket		Yoochoose (RC15)		RetailRocket	
	H@20	N@20	H@20	N@20	H@20	N@20	H@20	N@20
SASRec	0.561	0.317	0.355	0.222	0.560	0.315	0.351	0.218
SQN-SAS	0.612	0.341	0.430	0.271	0.581	0.327	0.423	0.263
EVAL-SAS	0.567	0.317	0.412	0.256	0.622	0.348	0.420	0.260
GRU	0.546	0.299	0.235	0.145	0.561	0.310	0.231	0.146
SQN-GRU	0.576	0.324	0.271	0.175	0.576	0.322	0.270	0.174
EVAL-GRU	0.576	0.320	0.269	0.172	0.578	0.323	0.272	0.176

Table 4.2: Reproduced Results against results reported in ICML'24 paper (Labarca et al., 2024) for *Purchase*-events for Yoochoose and RetailRocket datasets

Protocol- and Data-Related Observations Probably the most widely used evaluation protocol for session-based recommendation in the literature is the one that is used in the paper proposing the GRU4Rec model (Hidasi et al., 2016). In this protocol, which was also the basis for later comparative evaluations like (Ludewig

[1]https://www.acm.org/publications/policies/artifact-review-badging

et al., 2021), the data is temporally split into train, validation, and test data. A common choice is to use the last n days of data for testing, and some prior days for validation. For each session in the test set, the evaluation starts by revealing the first item in the session and then making predictions for the next item. Then, the next item is 'revealed' and the third item is predicted. This process is continued until the last item has been predicted.

In the analyzed papers (Labarca et al., 2024; Xin et al., 2020), two commonly used datasets for session-based recommendation are used, namely Yoochoose and Retailrocket. The data splitting in (Xin et al., 2020) however deviates from typical practices. Instead of using a temporal split—as to avoid data leakage by using future sessions to make predictions for earlier ones—a random subset of 200,000 sessions is sampled from the data. This subset is then randomly split into train, validation and test splits using an 8:1:1 ratio. Notably, while the recent work in (Labarca et al., 2024) largely follows the data-related procedures of (Xin et al., 2020), e.g., by using a 200,000 session subsample, data splitting according to (Labarca et al., 2024) is not done randomly but temporally. However, when analyzing the assumedly temporally ordered shared data splits of (Labarca et al., 2024), we found that the session IDs in the test split are the same as in the assumedly randomly sampled data splits shared in (Xin et al., 2020). Inquiries to the authors of (Labarca et al., 2024) about this phenomenon were without success.

In terms of specific details of the used evaluation protocol, further differences compared to typical evaluation setups of supervised models can be identified. First, given the splitting procedure, there can be items in the test data, which have not been seen in training. These are items, referred to as *unknown-items*, that users in the training set have not interacted with before. Nevertheless, during SQN training, the models are allowed to consider these *unknown-items*. This can be considered as a (weakly problematic) type of data leakage. In more typical evaluation setups, such items are removed from the test set. Since our evaluations showed that these *unknown-items* never appeared in top-k predictions, i.e., the model did not learn to predict such items in top positions, we see this issue as not too problematic.

A further difference with respect to more common evaluation protocols is that in the SQN evaluations, the model is also used to predict the first element in a session. We recall that in many other works, the first element is given as a starting point for the next-item prediction protocol.[2] Finally, a minor difference compared to influential earlier works like (Hidasi et al., 2016) lies in the choice of the evaluation metric, where (Labarca et al., 2024; Xin et al., 2020) report NDCG (Normalized Discounted Cumulative Gain) results rather MRR (Mean Reciprocal Rank) values.

[2]Clearly, making predictions for the first interaction in an anonymous session is relevant in practice, and may, for example, be based on general item popularity considerations.

All in all, the use of somewhat uncommon choices for the evaluation setup can make it difficult to make research results comparable across papers. In our present work, we therefore conduct experiments using different experimental configurations to analyze to what extent these configurations affect the experiment outcomes.

Code-Related Observations Both considered papers (Labarca et al., 2024; Xin et al., 2020) explore the value of adding additional heads to different supervised models like GRU4Rec or SASRec. An inspection of the provided code for these supervised models however showed that the used implementations deviate from the original papers. In the case of GRU4Rec and SASRec, for example not all aspects of the original proposals are implemented. If we look at the GRU4Rec original works, there are several unique strategies such as windowed-session fragments and sampling techniques, that are incorporated into their training process. We do not see these in the GRU baseline implementations in (Labarca et al., 2024; Xin et al., 2020).

Relying on proprietary or third-party implementations can be risky, as discussed by authors in (Hidasi & Czapp, 2023). Specifically, in (Hidasi & Czapp, 2023), the authors analyzed several implementations of the GRU4Rec models in different frameworks. They found that these implementations often strongly deviate from the original proposal, and that the performance of these alternative implementations can be substantially lower than for the original implementation. In our work, we therefore include the original GRU4Rec implementation as an additional baseline to analyze if there are any performance differences compared to the proprietary implementation.

A minor code-related aspect can be found in the data preprocessing script provided for (Labarca et al., 2024). As mentioned, this more recent work reports to use of a temporal data splitting procedure, and it provides both the raw and preprocessed datasets and a script to perform the splitting. Applying the script on the provided raw data however does not yield the provided preprocessed datasets.

4.3 Experimental Design

Given our observations in Section 4.2.2 regarding certain methodological choices in previous works, the goal of our present study is to reassess the effectiveness of RL-enhanced recommendations. To that purpose, we conducted a series of experiments involving different datasets and a suite of alternative baseline models.

Datasets, Pre-Processing and Data Splitting We consider both the two datasets from the original papers, as well as a third one (Diginetica), which is widely used in the literature. The inclusion of this additional dataset shall help us further

validate the generalizability of our findings. As mentioned, a certain inconsistency can be found in the literature regarding data preprocessing and data splitting. We conduct experiments following two alternative procedures:

- *SQN protocol*: This procedure is adopted in (Labarca et al., 2024), and we use it for exact reproducibility. In this approach, sessions and items with less than 3 interactions are removed. The sessions of all the datasets are *temporally* split into train, validation and test following an 8:1:1 ratio. We consider both click and purchase data, except for the Diginetica dataset, where only clicks are available. We recall that the data splits shared by (Labarca et al., 2024) may not have been temporally ordered as described in the paper. We have therefore implemented a script for data splitting, following the procedures of (Labarca et al., 2024) to create the splits by our own. Thus, the experimental results reported below are based on our own data splits. Table 4.4 shows the statistics of the pre-processed datasets.

Dataset		*#events*	*#sessions*	*#items*	*#n days*
Yoochoose (1/64)	train	435,891	106,118	17,106	3
	valid	58,233	12,372	6,359	1
	test	70,641	15,234	6,367	1
Retailrocket	train	995,944	280,115	48,728	124
	valid	48,113	14,323	15,401	7
	test	40,520	12,181	13,643	7
Diginetica	train	859,434	176,975	43,023	138
	valid	56,930	11,834	18,410	7
	test	76,901	15,967	21,146	7

Table 4.3: Dataset characteristics after applying *GRU-protocol* procedures.

- *GRU4Rec protocol*: Following common practice in the literature (Hidasi et al., 2016), sessions with only one interaction and items with less than 5 interactions are removed. The sessions of the last n days are used for testing. A validation set is constructed from the training data in the same fashion. As also common in the literature (J. Li et al., 2017; Q. Liu et al., 2018), only the most recent interactions (1/64 of the data) are considered for the large Yoochoose dataset. Furthermore, we only consider *click* and *view* events in our experiments. Table 4.3 shows the statistics of the pre-processed datasets.

Compared Models In our experiments, we benchmark central models from (Labarca et al., 2024) and (Xin et al., 2020) against *(i)* two neural session-based models of that

		#events	#sessions	#items	#clicks	#purchases
Yoochoose (RC15)	train	918,122	160,000	24,408	884,315	33,807
	valid	115,679	20,000	10,131	110,761	4,918
	test	97,086	17,897	8,987	93,079	4,007
RetailRocket	train	933,690	156,419	67,105	890,937	42,753
	valid	122,446	19,552	31,984	116,722	5,724
	test	169,615	18,446	35,975	161,118	8,497
Diginetica	train	763,732	129,250	57,667	763,732	-
	valid	93,808	16,155	27,667	93,808	-
	test	93,848	15,910	26,350	93,848	-

Table 4.4: Dataset characteristics after applying *SQN-protocol* procedures.

time, GRU4Rec based on its original implementation and NARM, and *(ii)* against two models based on nearest neighbours, SKNN and VSKNN (Ludewig et al., 2019). The overall set of compared models is shown in Table 4.5.

Neural and Non-Neural Baselines	
SKNN (Ludewig et al., 2019)	A basic session-based k-nearest neighbour technique in which every item in a session is assumed to be equally important when computing similarities.
VSKNN (Ludewig et al., 2019)	Similar to S-KNN, but based on a similarity function that puts more emphasis on the more recent events in a session.
GRU4Rec (Hidasi & Karatzoglou, 2018; Hidasi et al., 2016)	The first neural approach using Recurrent Neural Networks (RNNs) for session-based recommendation. The technique was later on improved by more effective loss functions (Hidasi & Karatzoglou, 2018).
NARM (J. Li et al., 2017)	This model extends GRU4Rec with improved session modelling using an attention mechanism.
Baseline implementations from (Labarca et al., 2024; Xin et al., 2020)	
GRU (RL)	GRU baseline implementation from (Labarca et al., 2024; Xin et al., 2020), with authors referring to GRU4Rec (Hidasi et al., 2016).
SAS (RL)	Self-attention based baseline implementation from (Labarca et al., 2024; Xin et al., 2020), with the authors referring to SASRec (Kang & McAuley, 2018).
RL-enhanced models	
GRU-SQN, SAS-SQN	Self-supervised Q-Learning models from (Xin et al., 2020), combined with GRU and SAS, respectively.
Auxiliary-loss based models	
GRU-EVAL SAS-EVAL	Variation of SQN proposed in (Labarca et al., 2024), using an alternative loss function for next-item modeling.

Table 4.5: Compared Models

Evaluation Protocol Following (Hidasi et al., 2016) and subsequent works, the evaluation procedure looks at the *immediate next item* predicted by the RS. The items of a session are revealed one-by-one and we then check if the next item is seen in a ranked list of k items. For each session, the first item is always provided and the models are tested on the remaining events. The metrics used in the evaluations are Recall@k, Mean Reciprocal Rank (MRR@k) and Normalized Discounted Cumulative Gain (NDCG@k).

Implementation and Hyperparameter Tuning For the SKNN and VSKNN methods, we used the implementations from the *session-rec*[3] framework. For GRU4-Rec, we relied on the official PyTorch implementation[4] by the authors. NARM is based on a publicly available PyTorch version[5], which we verified to be reproducible. Since the implementations of the SQN variants were based on TensorFlow 1.x, we could not run them on newer GPU hardware, but used CPUs instead.

We systematically tuned the hyperparameters of all models for all three datasets and for both evaluation protocols separately. We used a random-search strategy with 100 iterations, with MRR@20 set as an optimization target. During this process, the models are evaluated only on the validation data. The considered hyperparameter ranges and optimal values are documented in the online repository[6]. Once the optimal parameters are determined, the methods are then trained and evaluation is done on the test data. We followed the procedures from the original papers of each method as closely as possible. For the KNN variants and GRU4Rec, the final training is done using the full training interactions, which is a combination of both train and validation splits. For the NARM and SQN methods, the models are optimized on the train data split, and the final model for testing is picked based on the best MRR@20 for the validation set at each epoch. The selected model is then finally trained on the union of the training and validation data and evaluated on the test data.

4.4 Results

In this section, we first discuss the results when applying the evaluation protocol from (Labarca et al., 2024) (termed *SQN-protocol*) as described above. Afterwards, we benchmark the different models under the more common *GRU4Rec* protocol.

[3]https://github.com/rn5l/session-rec
[4]https://github.com/hidasib/GRU4Rec_PyTorch_Official
[5]https://github.com/Wang-Shuo/Neural-Attentive-Session-Based-Recommendation-PyTorch
[6]https://dilina-r.github.io/rl-rec/

	Yoochoose (RC15)			Retailrocket			Diginetica		
	H@20	N@20	M@20	H@20	N@20	M@20	H@20	N@20	M@20
SKNN	0.542	0.308	0.239	0.399	0.265	0.225	0.465	0.229	0.162
VSKNN	0.580	0.335	0.262	0.454	0.328	0.291	0.465	0.227	0.160
GRU4Rec	**0.617**	**0.356**	**0.278**	**0.480**	**0.341**	**0.300**	**0.504**	**0.243**	**0.169**
NARM	0.562	0.314	0.240	0.418	0.313	0.281	0.459	0.226	0.160
SAS (RL)	0.551	0.305	0.233	0.403	0.263	0.221	0.383	0.173	0.114
SAS-SQN	0.560	0.309	0.235	0.398	0.267	0.228	0.386	0.175	0.116
SAS-EVAL	0.560	0.312	0.239	0.399	0.266	0.227	0.389	0.176	0.116
GRU (RL)	0.463	0.254	0.193	0.218	0.138	0.115	0.225	0.100	0.066
GRU-SQN	0.473	0.259	0.196	0.271	0.171	0.141	0.257	0.113	0.076
GRU-EVAL	0.476	0.261	0.198	0.268	0.170	0.141	0.256	0.113	0.075

Table 4.6: Results using the temporal *SQN protocol*, using only *view* events

	Yoochoose (RC15)			Retailrocket		
	H@20	N@20	M@20	H@20	N@20	M@20
SKNN	**0.807**	**0.511**	**0.421**	0.503	0.352	0.307
VSKNN	0.707	0.426	0.342	0.752	0.659	0.631
GRU4Rec	0.537	0.344	0.286	**0.824**	**0.755**	**0.734**
NARM	0.604	0.349	0.274	0.692	0.575	0.540
SAS (RL)	0.474	0.279	0.222	0.725	0.571	0.524
SAS-SQN	0.475	0.280	0.222	0.721	0.599	0.561
SAS-EVAL	0.476	0.287	0.230	0.723	0.596	0.556
GRU (RL)	0.425	0.232	0.175	0.254	0.162	0.135
GRU-SQN	0.444	0.244	0.186	0.331	0.213	0.178
GRU-EVAL	0.445	0.247	0.189	0.330	0.209	0.173

Table 4.7: Results using the temporal *SQN protocol*, using *purchase* events

4.4.1 Results using the *SQN protocol*

We recall that in this experiment we rely on the SQN evaluation protocol, but use temporal data splits which we created by ourselves, following the descriptions in (Labarca et al., 2024). In Table 4.6, we show the results when considering *item view* events. Table 4.7 reports the numbers obtained when considering *purchase* events. We note that the Diginetica dataset does not contain purchase events. We report the metric values for a cutoff length of 20, as done, e.g., in (Hidasi et al., 2016). The results for other cutoff lengths show the same patterns and can be found in the Appendix 6.1.

Our main observations can be summarized as follows.

No consistent gain of enhancing baselines We recall that we *could* reproduce the effects reported in (Labarca et al., 2024; Xin et al., 2020) regarding the improved performance when adding an RL-component (SQN, (Xin et al., 2020)) or an additional loss (EVAL, (Labarca et al., 2024)) to the GRU and SAS baseline

models when using the shared dataset and code artifacts. For the sake of brevity, we refer to the SQN and EVAL models as "RL-based models" from here on. When using a proper *temporal split*, however, we obtain mixed results. Specifically, we find that marked performance improvements are only observed for the Retailrocket and Diginetica datasets. For the Yoochoose dataset, instead, we found that extending the GRU (RL) baseline did not lead to a performance improvement, and only to a tiny increase when SAS (RL) is used as a baseline. The trend is the same for measurements based on item view and purchase data.

Better performance of original GRU4Rec Across all datasets and measurements, we found that a well-tuned GRU4Rec model, when based on the original implementation, outperforms the RL-based methods and the underlying GRU (RL) and SAS (RL) baselines. The obtained metric values are also higher than those reported in (Labarca et al., 2024; Xin et al., 2020) when using the random splits. Overall, while some improvements over the baselines can be observed when adding an RL component or an auxiliary loss, these improvements are obtained when applied to what appear to be comparably weak baseline models. Besides differences in the GRU (RL) and GRU4Rec implementations, one additional reason for the observed performance differences between these models can lie in the hyperparameter tuning of the GRU (RL) and SAS (RL) models in the original papers. In (Xin et al., 2020), it is stated that the item embedding size—a central hyperparameter—was set to a fixed size for all models without further tuning. Other hyperparameters were probably not systematically tuned for each dataset either.

Regarding the other additional models in our comparison, we find that the VSKNN model consistently performs well, but usually is still markedly less effective on these datasets than GRU4Rec. The relative good performance of VSKNN might be due to the algorithm's strategy to search for neighbors within the most recent sessions in the data. The NARM model, to some surprise, is only competitive on the Diginetica dataset. While it consistently outperforms the RL-based models, the performance of NARM is often markedly lower than the performance of GRU4Rec.

We note that we observe much higher *absolute* metric values for purchase events compared to what is reported in (Labarca et al., 2024) and (Xin et al., 2020). We recall that the Yoochoose data splits generated in our experiments are not identical to those of these prior works, as we had to create the data splits on our own, as discussed above. As such, different results in terms of absolute values can be expected in general. However, we speculate that our systematic hyperparameter tuning may have largely contributed to this improved performance. In terms of ranking the algorithms, we observe the best results again from GRU4Rec and the KNN variants, even though they were not provided with information about different

interaction types.

4.4.2 Results using the *GRU4Rec protocol*

The results when applying the most common evaluation protocol for session-based recommendation are shown in Table 4.8 (view events). Our main observations are as follows.

No consistent gain of enhancing baselines Like in the SQN-protocol evaluation, we find that adding an RL component or an auxiliary loss helps to improve the metrics in many but not all cases. For the Yoochoose and Diginetica datasets, we do not see any notable improvement over the SAS (RL) baseline from the respective RL components. However consistent performance gains over the GRU (RL) baseline can be seen by GRU-SQN and GRU-EVAL across all datasets. For the remaining scenarios, small accuracy improvements, as reported in (Labarca et al., 2024; Xin et al., 2020), can be confirmed.

Best performance obtained by GRU4Rec In overall, GRU4Rec turns out to be again the best-performing model, except for the Yoochoose dataset where NARM shows a slightly higher HitRate@20. The NARM model works generally well when using the GRU4Rec evaluation protocol, which is also in line with previous comparative evaluations (Ludewig et al., 2021). For the Yoochoose dataset, we observe that the SAS (RL) baseline and its enhanced models reach performance levels that are closer to those of NARM and GRU4Rec. For the other datasets, a marked gap however remains between the SAS (RL) model and the best-performing ones. KNN approaches, and in particular VS-KNN, show performance results that are often competitive with the neural approaches for the Yoochoose and Retailrocket datasets.

Overall, we find similar patterns for both evaluation schemes, i.e., the *SQN protocol* and the *GRU4Rec protocol*. Specifically, we observed that a well-tuned neural model like GRU4Rec is favourable in terms of performance results compared to the RL-enhanced models proposed in (Labarca et al., 2024; Xin et al., 2020).

4.4.3 Effects of Dismissing Validation Data

When reproducing the results of (Labarca et al., 2024; Xin et al., 2020), we noticed that in both papers the data of the validation split, i.e., 10 percent of the data, was *not* used for training the final model that is used for measuring the model's performance on the test set. We recall that validation splits are commonly used to

	Yoochoose			Retailrocket			Diginetica		
	H@20	N@20	M@20	H@20	N@20	M@20	H@20	N@20	M@20
SKNN	0.646	0.341	0.253	0.593	0.400	0.342	0.502	0.257	0.188
VSKNN	0.699	0.382	0.289	0.596	**0.403**	**0.345**	0.513	0.319	0.263
GRU4Rec	0.713	**0.402**	**0.310**	**0.610**	0.369	0.299	**0.655**	**0.394**	**0.317**
NARM	**0.715**	**0.402**	**0.310**	0.556	0.389	0.340	0.628	0.364	0.287
SAS (RL)	0.700	0.393	0.302	0.506	0.310	0.253	0.572	0.320	0.247
SAS-SQN	0.703	0.394	0.302	0.519	0.321	0.263	0.575	0.324	0.251
SAS-EVAL	0.704	0.394	0.302	0.523	0.324	0.266	0.571	0.322	0.248
GRU (RL)	0.622	0.345	0.264	0.433	0.272	0.224	0.376	0.203	0.152
GRU-SQN	0.643	0.356	0.272	0.455	0.288	0.239	0.467	0.248	0.184
GRU-EVAL	0.647	0.358	0.273	0.454	0.289	0.240	0.469	0.248	0.184

Table 4.8: Results using the *GRU4Rec protocol*, using only *view* events

check how well a trained model generalizes to unseen data and to avoid overfitting. In our experiments reported above, we followed common procedures and used the validation data in our experiments to determine optimal hyperparameters for each model and dataset. Once these hyperparameters were determined, we used the union of the training and the validation data to train the final model.

	Yoochoose (clicks)		Retailrocket (clicks)	
	H@20	M@20	H@20	M@20
NARM	0.562	0.240	0.418	0.281
SAS-SQN	0.560	0.235	0.398	0.228
GRU-SQN	0.473	0.196	0.271	0.141
NARM (*w/o)	0.308	0.147	0.328	0.275
SAS-SQN (*w/o)	0.310	0.130	0.361	0.212
GRU-SQN (*w/o)	0.260	0.115	0.263	0.133

Table 4.9: Results for SQN Protocol Yoochoose and Retailrocket (clicks only) data trained using with and without (*w/o) validation data for NARM, GRU-SQN and SAS-SQN

Dismissing the validation data in this final training step not only leads to a more sparse data situation, it also has the effect that the model is trained on older data. Intuitively, this can be very disadvantageous in sequential or time-based evaluation scenarios, as more recent data points are ignored. The effects of only using the training data (without the validation data) are illustrated in Table 4.9. In the table, we first show the performance results when the union of training and validation data is used to train the final model. Below, marked with "*w/o", the results are presented when the validation data is not considered. The results show that the performance drop can be substantial (e.g., about 40%) for the considered models.

We recall that we could reproduce the reported numbers from the original pa-

pers (Labarca et al., 2024; Xin et al., 2020), as shown in Table 4.1 and Table 4.2. These reproduced performance numbers are however much lower than those we report for the RL-enhanced models and for NARM in our own experiments above. The reason is that in our experiments we considered the union of the training and validation data for the final training step to ensure a fair comparison. Besides the fact that the performance numbers are higher in this comparison, what can be observed is that the value of enhancing the baseline models like SAS (RL) with an additional RL component or loss function starts to diminish when more data is used for training. Overall, the non-consideration of the validation data remains an unclear methodological choice in the examined prior works.

4.5 Discussion

To begin with, it is highly commendable that the authors of (Labarca et al., 2024; Xin et al., 2020) share the necessary artifacts for reproducibility. While researchers increasingly provide reproducibility artifacts, this is still far from common practice, despite the existence of various reproducibility guidelines provided by journals and conferences. The provided artifacts allowed us—after some modifications—to quite closely reproduce the results reported in the respective papers. The analyses reported in our present paper however raise a number of open issues.

Value of RL for Next-Item-Prediction Tasks The reproduced results confirm that both the RL-based proposal from (Xin et al., 2020) as well as the model based on an auxiliary loss function (Labarca et al., 2024) can lead to increased performance over the chosen baselines, even when applying a next-item prediction evaluation protocol. However, as discussed in detail in (Labarca et al., 2024), it is not entirely clear if the gains reported in (Xin et al., 2020) truly stem from the "reinforcement learning nature" of their approach. Our present work furthermore shows that the obtained improvements in (Labarca et al., 2024; Xin et al., 2020) are obtained over baseline models that are not competitive with the state of the art, i.e., GRU4Rec in our case. It therefore remains to show if the proposals from (Labarca et al., 2024; Xin et al., 2020) would lead to increased performance when applied over more recent models. An investigation of this question is left for future work. Generally, we emphasize that our findings in no way rule out that RL techniques can be an important performance driver for recommender systems. Our present study mainly underlines the questions and concerns raised in (Labarca et al., 2024) and (Deffayet et al., 2023) regarding the unexpected effectiveness of RL techniques when applying a myopic evaluation protocol. As indicated in (Labarca et al., 2024), while the RL component approach from (Xin et al., 2020) can lead to performance improvements,

it is not entirely clear if the performance derives from what the model is assumed to learn, or if the additional RL component mainly increases the modelling capacity of the approach. A related discussion can be found in (Ferrari Dacrema et al., 2020). In this latter work, it was shown that a recommendation approach based on CNNs actually does not learn what it is assumed to do, but that the used CNN layer rather acts as a universal approximator, which still leads to increased performance results.

Factors Potentially Hampering Progress A main question arising from our analyses is if true progress is achieved through the proposed models, given that GRU4Rec alone leads to superior performance in our study. Similar questions were raised earlier in the general area of recommender systems. In various of these earlier works, a number of factors were identified that may contribute to what is referred to as "phantom progress" (Ferrari Dacrema et al., 2021). One main factor in this context lies both in the choice of the baselines and the efforts that were put into tuning the baseline models. Already in 2009, a related study from the field of information retrieval indicated that the reported improvements in the literature often "*don't add up.*"(Armstrong et al., 2009).[7] Similar phenomena were reported in recent years in various papers, which found that even very recent and sometimes computationally complex models are often outperformed by long-existing baselines techniques when these baselines are properly tuned (Anelli et al., 2023; Ferrari Dacrema et al., 2019; Ludewig et al., 2021; Milogradskii et al., 2024; Rendle et al., 2020, 2022; Shehzad & Jannach, 2023). Besides the lack of proper tuning of the baselines, another phenomenon found in the literature is called "propagation of weak baselines" (Ferrari Dacrema et al., 2021). This phenomenon may occur when the set of chosen baselines only includes models from one particular family of algorithms, e.g., of recent neural models. In the context of this present study, the proposed models (Labarca et al., 2024; Xin et al., 2020) were applied over state-of-the-art sequential approaches, namely GRU4Rec and SASRec. As discussed earlier, however, it is not entirely clear if these baselines were systematically tuned to their full capacity. Furthermore, alternative implementations of the baselines were used in the examined work, an issue that was recently discussed in (Hidasi & Czapp, 2023), where it was found that third-party implementations of GRU4Rec often exhibit substantially lower performance than the original implementation. Ultimately, if future works build on the results presented in (Labarca et al., 2024; Xin et al., 2020), the danger arises that even if a substantial gain over these results is reported, it remains unclear if progress over a well-tuned GRU4Rec baseline is actually achieved.

Another factor that makes it difficult to assess the progress achieved through a new model is that researchers have a certain degree of freedom when it comes

[7] See also (Lin, 2019).

to the details of the evaluation procedure (Ferrari Dacrema et al., 2021). Such inconsistencies in the literature make it difficult to compare the results reported in different papers. In the context of our present study, we observed that data splitting procedures were applied that might be considered not the most frequent ones in the literature on session-based or sequential recommendation.

Research Limitations and Ways Forward Our work so far focused on re-assessing the effectiveness of recommendation models reported in (Xin et al., 2020) by comparing them to well-tuned state-of-the-art sequential models. The combination of these well-tuned baselines, in particular the original implementation of GRU4Rec, with RL components or auxiliary loss functions as proposed in (Labarca et al., 2024; Xin et al., 2020) is thus out of the scope of this present work. In terms of research limitations, our present study has focused on two particular research works. Therefore, we cannot draw more generalizable conclusions about the broader class of RL-based or RL-enhanced recommender systems.

On a more general level, however, our work supports previous concerns about the suitability of using a next-item prediction evaluation protocol when it comes to evaluating RL-based approaches to recommender systems, as such approaches commonly aim to maximize the long-term reward of the recommendations (Deffayet et al., 2023; Labarca et al., 2024). We note that discussions surrounding the topic of reliable recommender systems evaluation are not limited to reinforcement learning based approaches, and it is becoming a rising concern in the field. Offline evaluations have become a standard practice in the literature, even though it is not entirely clear if the performance gains obtained in a typical "hide-and-predict" offline evaluation scheme would translate to more effective systems in practice (Cremonesi & Jannach, 2021; Gomez-Uribe & Hunt, 2015).

Given the particularities of reinforcement learning based approaches, a growing trend and promising direction in the related literature is to utilize *simulation* environments both for training and model evaluation (Afsar et al., 2022; S. Wang et al., 2023). Examples of such recommender systems simulators include RecoGym (Rohde et al., 2018), RecSim (Ie et al., 2019) or KuaiSim(K. Zhao, Liu, et al., 2023). We believe that simulations represent a promising way for reliable offline evaluation of longitudinal effects of RL-based models and beyond (Adomavicius et al., 2021). So far, however, a unified framework and standard benchmark protocol for such simulation-based evaluation approaches has not been established yet in the research community. In this context, we believe it is highly important that such an evaluation protocol should go beyond accuracy and (long-term) reward measures and include additional metrics to assess the novelty, diversity, and popularity of the generated recommendations (Kaminskas & Bridge, 2016; Klimashevskaia et al., 2024).

This is particularly important in our view given the intrinsic exploratory nature of RL-based models.

4.6 Conclusion

Inspired by a recent work on the unexpected effectiveness of reinforcement learning based approaches for next-item prediction tasks, we have conducted a series of experiments to reassess previous findings from the literature. While we could reproduce the results reported in earlier papers, our experiments indicate the performance of the models proposed in these previous papers is actually lower than what can be achieved when using a plain GRU4Rec model or models based on nearest-neighbours. Our analysis therefore provides additional indications that we as a research community have to further increase methodological rigor to ensure constant and reliable progress in our field.

Chapter 5

Conclusion and Future directions

5.1 Summary

In Chapter 2, we revisited the user-cold start problem in Recommender Systems. We proposed a novel Monte-Carlo Tree Search (MCTS) based recommender that quickly learns about a new user's preferences in an online manner. In this approach, we consider a setup where users are clustered into groups based on the similarities in their preferences and users in the same group share similar traits and preferences at a higher degree. By correctly estimating the group, the RS can then recommend items based on the known preferences within the user-group. We show that, when a new user arrives, the MCTS quickly learns about the group to which the user belongs to. The MCTS does this, by selecting items that will allow it to quickly learn about the user with fewer item interactions. In our experimental results, we showed that the cold-start performance of the MCTS surpasses that of an optimized Decision Tree and state-of-the-art Cluster-based Bandits (Shams et al., 2021). The MCTS achieved this recommendation performance while being computationally lightweight. We also showed that the MCTS can handle different feedback apart from explicit ratings, mainly pairwise-comparison (i.e.: the user has to select from two suggested items) and missing-not-at-random (i.e.: a user may choose not to provide some ratings) cases.

In Chapter 3, we introduced a new sequential transformer architecture *RLT4Rec* and demonstrated its performance in a range of item recommendation tasks. RLT4Rec handles new users and established users within the same consistent framework, it does not require a separate cold-start mode. RLT4Rec takes as input a sequence of user (item, rating) pairs and outputs the next item to present to the user, and unlike other RL approaches, it does not require input of a state observation or estimate. In our experiments, we measured the performance of our RLT4Rec in providing recommendations to both new users (i.e. user-cold start scenario) and established users.

Our findings showed that the RLT4Rec consistently improves upon the performance of MCTS (proposed in Chapter 3) for new users, and surpasses the RL baselines PRL(Xin, Pimentel, et al., 2022) and Decision Transformer(L. Chen et al., 2021) for established users. It was shown that RLT4Rec learns to generate "good" sequences of items (i.e. users tend to provide high ratings for these items) even when trained on random sub-optimal training sequences. We also showed that RLT4Rec effectively learns an internal user representation despite not having a separate state-estimation component. In another set of independent experiments, we show that with a "good" state estimation technique, even a basic Neural Network can be employed for effective recommendations while requiring minimal computational power.

In Chapter 4, we first looked at the reproducibility of the RL-based recommenders in literature and noticed a severe lack of reproducibility, confirming the findings of (Cavenaghi et al., 2023). Out of the literature, we found the *SQN framework* from (Xin et al., 2020) to be reproducible and has been used as a backbone for many RL-based RS literature in recommender systems. Inspired by a recent study by (Labarca et al., 2024), we explored the effectiveness of these RL-based RS in sequential recommendation, primarily with the SQN framework. However, we encountered numerous inconsistencies in this literature, making it difficult to observe the true performance gain of these RL-enhanced models against other sequential recommenders in the literature. Therefore, through methodical evaluations, we analyzed the performance of the RL-enhanced models in SQN framework against several supervised learning-based sequential recommender models in myopic *next-item* prediction tasks. Our findings showed that these RL-enhanced methods in the SQN framework show no noticeable gain over supervised methods, such as GRU4Rec and nearest-neighbours methods.

5.2 Future Work

5.2.1 Improvements to MCTS

Offline Training

In the MCTS approach proposed in Chapter 2 a new lookahead subtree is constructed at each step t in order to select the next item to ask a new user to rate. While our measurements show that this tree can be constructed rather quickly (except when ranked comparison feedback is used), a further computational saving might be possible by storing the generated lookahead subtrees for new users and using these as training data for a neural net. That is, it might be possible to train a neural net offline to capture the decisions made by the MCTS approach and then use this neural net to make online predictions. However, we leave the investigation

of this interesting topic to future work.

Item-cold start with MCTS

In Chapter 2, we show that our MCTS quickly learns a user's preferences from each item interaction and feedback received. Similarly, it can be extended to learn about items, in the sense these feedback from users can be used to update the item distributions. A recommenders system typically has a large number of items and many of them have a relatively lower number of interactions with users (i.e. due to the highly sparse nature of recommender datasets). This can lead to poor item representations and lower chances of being recommended to users. In the case of MCTS, having fewer item ratings can result in noisy item-rating distributions. At the same time, there can be new items added to the system that are referred to as cold-start items.

We can use the same MCTS framework to ask established users (i.e. users who have interacted with a higher number of items) to rate these lesser-known items. This can be done intelligently, such that users are carefully selected to rate particular items. The majority of Collaborative-filtering techniques require re-training of models or batch updates to learn newer item representations. However with our MCTS setup, the item-distributions can be learned in an online manner. This way, the MCTS can handle both user and item cold start in the same framework.

5.2.2 Handling different types of input

Typically a recommender system will have access to additional information about the users and items. For example, the country/city they are located in, user gender and demographics, and data from other services used by the user. This supplementary information can be used as inputs to the RS, providing contextual information that can be used to improve recommendations. We show that our MCTS and RLT4Rec can be extended to accommodate these types of inputs.

MCTS: The proposed MCTS-based approach can be directly extended to incorporate such contextual information. For example, users sharing the same context (e.g. located in the same country) can be gathered together, the number of user groups and the item rating means and variances for each group estimated and then the MCTS-based cold-start approach applied to this sub-population. This is similar to the approach used in contextual bandits. Contextual information can also be used to adjust the prior probabilities $p_g^{(0)}, g \in \mathcal{G}$ of the group to which the user is initially estimated to belong.

RLT4Rec: Our proposed RLT4Rec model is inherently capable of incorporating different types of inputs. For example, the additional information can be appended

to the input sequence provided to the transformer architecture by appending them as tokens in the relevant temporal sequence. RLT4Rec is also capable of accommodating multi-modal data (eg: textual reviews/descriptions, visual images of items, etc). To incorporate these, independent feature extraction components will be required. For instance, convolutional neural networks (CNNs) can be employed to extract visual features from item images, while language models such as BERT (Devlin et al., 2019) can be utilized to derive contextual embeddings from textual content. The extracted feature embeddings can then be included in the input sequence to the RLT4Rec.

Chapter 6

Appendix

6.1 Appendix A

This appendix section contains additional experimental results for Chapter 4.

6.1.1 Results for GRU-protocol with different cut-off lengths

Tables 6.1-6.3 show the measured HitRate and Mean Reciprocal Rank (MRR) at cut-off length 5,10,20 for the Yoochoose, RetailRocket and Diginetica datasets with GRU Protocol.

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.395	0.226	0.527	0.244	0.646	0.253
VSKNN	0.460	0.263	0.592	0.281	0.699	0.289
GRU4Rec	**0.477**	**0.285**	**0.609**	**0.303**	0.713	**0.310**
NARM	0.475	**0.285**	**0.609**	**0.303**	**0.715**	**0.310**
SAS(RL)	0.466	0.278	0.595	0.295	0.700	0.302
SAS-SQN	0.466	0.277	0.596	0.295	0.703	0.302
SAS-EVAL	0.467	0.277	0.598	0.295	0.704	0.302
GRU(RL)	0.402	0.241	0.520	0.257	0.622	0.264
GRU-SQN	0.417	0.248	0.539	0.265	0.643	0.272
GRU-EVAL	0.419	0.249	0.544	0.266	0.647	0.273

Table 6.1: Measured HitRate, MRR for Yoochoose dataset using the GRU-protocol at cut-off lengths 5, 10 and 20

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.462	0.328	0.535	0.338	0.593	0.342
VSKNN	**0.466**	**0.331**	**0.537**	**0.341**	0.596	**0.345**
GRU4Rec	0.419	0.279	0.518	0.292	**0.610**	0.299
NARM	0.438	0.327	0.499	0.336	0.556	0.340
SAS(RL)	0.358	0.237	0.437	0.248	0.506	0.253
SAS-SQN	0.367	0.247	0.447	0.258	0.519	0.263
SAS-EVAL	0.372	0.250	0.453	0.261	0.523	0.266
GRU(RL)	0.317	0.211	0.379	0.220	0.433	0.224
GRU-SQN	0.332	0.226	0.398	0.235	0.455	0.239
GRU-EVAL	0.336	0.228	0.397	0.237	0.454	0.240

Table 6.2: Measured HitRate, MRR for RetailRocket dataset using the GRU-protocol at cut-off lengths 5, 10 and 20

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.278	0.166	0.386	0.180	0.502	0.188
VSKNN	0.334	0.245	0.419	0.257	0.513	0.263
GRU4Rec	**0.453**	**0.296**	**0.556**	**0.310**	**0.655**	**0.317**
NARM	0.415	0.265	0.524	0.280	0.628	0.287
SAS(RL)	0.373	0.226	0.477	0.240	0.572	0.247
SAS-SQN	0.377	0.230	0.482	0.244	0.581	0.251
SAS-EVAL	0.374	0.227	0.478	0.241	0.576	0.248
GRU(RL)	0.232	0.138	0.305	0.148	0.376	0.152
GRU-SQN	0.281	0.165	0.379	0.178	0.467	0.184
GRU-EVAL	0.284	0.165	0.379	0.178	0.469	0.184

Table 6.3: Measured HitRate, MRR for Diginetica dataset using the GRU-protocol at cut-off lengths 5, 10 and 20

6.1.2 Results for SQN-protocol with different cut-off lengths

Results for Clicks events

Tables 6.4-6.6 show the measured HitRate and MRR for clicks-events at cut-off length 5,10,20 for the Yoochoose, RetailRocket and Diginetica datasets with GRU Protocol.

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.362	0.220	0.462	0.233	0.542	0.239
VSKNN	0.402	0.243	0.498	0.256	0.580	0.262
GRU4Rec	**0.426**	**0.258**	**0.533**	**0.272**	**0.617**	**0.278**
NARM	0.373	0.220	0.478	0.235	0.562	0.240
SAS(RL)	0.359	0.212	0.464	0.226	0.551	0.233
SAS-SQN	0.367	0.215	0.469	0.228	0.560	0.235
SAS-EVAL	0.366	0.218	0.474	0.233	0.560	0.239
GRU(RL)	0.300	0.176	0.390	0.188	0.463	0.193
GRU-SQN	0.305	0.178	0.396	0.191	0.473	0.196
GRU-EVAL	0.307	0.180	0.400	0.192	0.476	0.198

Table 6.4: Measured HitRate, MRR for Yoochoose *clicks* events, using temporal SQN-protocol at cut-off lengths 5, 10 and 20

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.303	0.215	0.355	0.222	0.399	0.225
VSKNN	0.368	0.282	0.416	0.288	0.454	0.291
GRU4Rec	**0.378**	**0.290**	**0.431**	**0.297**	**0.480**	**0.300**
NARM	0.343	0.274	0.381	0.279	0.418	0.281
SAS(RL)	0.293	0.210	0.350	0.218	0.403	0.221
SAS-SQN	0.296	0.218	0.348	0.225	0.398	0.228
SAS-EVAL	0.296	0.216	0.349	0.223	0.399	0.227
GRU(RL)	0.156	0.108	0.187	0.113	0.218	0.115
GRU-SQN	0.196	0.134	0.235	0.139	0.271	0.141
GRU-EVAL	0.192	0.134	0.232	0.139	0.268	0.141

Table 6.5: Measured HitRate, MRR for RetailRocket *clicks* events, using temporal SQN-protocol at cut-off lengths 5, 10 and 20

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.251	0.140	0.354	0.154	0.465	0.162
VSKNN	0.247	0.139	0.349	0.152	0.465	0.160
GRU4Rec	**0.261**	**0.145**	**0.376**	**0.160**	**0.504**	**0.169**
NARM	0.242	0.139	0.342	0.152	0.459	0.160
SAS(RL)	0.175	0.094	0.269	0.106	0.383	0.114
SAS-SQN	0.178	0.096	0.271	0.108	0.386	0.116
SAS-EVAL	0.178	0.096	0.271	0.108	0.389	0.116
GRU(RL)	0.103	0.055	0.157	0.062	0.225	0.067
GRU-SQN	0.120	0.064	0.182	0.072	0.257	0.077
GRU-EVAL	0.120	0.064	0.182	0.072	0.256	0.077

Table 6.6: Measured HitRate, MRR for Diginetica *clicks* events, using temporal SQN-protocol at cut-off lengths 5, 10 and 20

Results for Purchase events

Below are the measured HitRate and MRR for purchase-events at cut-off length 5,10,20 for the Yoochoose (Table 6.7) and RetailRocket (Table 6.8) datasets with SQN Protocol.

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	**0.619**	**0.400**	**0.739**	**0.416**	**0.807**	**0.421**
VSKNN	0.513	0.322	0.616	0.336	0.707	0.342
GRU4Rec	0.394	0.271	0.476	0.282	0.537	0.286
NARM	0.413	0.254	0.521	0.268	0.604	0.274
SAS(RL)	0.326	0.206	0.411	0.217	0.474	0.222
SAS-SQN	0.329	0.207	0.407	0.218	0.475	0.222
SAS-EVAL	0.335	0.216	0.412	0.226	0.476	0.230
GRU(RL)	0.270	0.159	0.357	0.170	0.425	0.175
GRU-SQN	0.286	0.170	0.366	0.181	0.444	0.186
GRU-EVAL	0.285	0.172	0.373	0.184	0.445	0.189

Table 6.7: Measured HitRate, MRR for Yoochoose *purchase* data using temporal SQN-protocol at cut-off lengths 5, 10 and 20

	H@5	M@5	H@10	M@10	H@20	M@20
SKNN	0.399	0.296	0.455	0.303	0.503	0.307
VSKNN	0.700	0.625	0.728	0.629	0.752	0.631
GRU4Rec	**0.781**	**0.729**	**0.806**	**0.733**	**0.824**	**0.734**
NARM	0.623	0.532	0.659	0.537	0.692	0.540
SAS(RL)	0.634	0.514	0.689	0.522	0.725	0.524
SAS-SQN	0.655	0.553	0.695	0.559	0.721	0.561
SAS-EVAL	0.657	0.549	0.695	0.554	0.723	0.556
GRU(RL)	0.180	0.127	0.216	0.132	0.254	0.135
GRU-SQN	0.238	0.168	0.285	0.175	0.331	0.178
GRU-EVAL	0.236	0.164	0.282	0.170	0.330	0.173

Table 6.8: Measured HitRate, MRR for RetailRocket *purchase* data using temporal SQN-protocol at cut-off lengths 5, 10 and 20

Bibliography

Abdullah, N. A., Rasheed, R. A., Nasir, M. H. N. M., & Rahman, M. M. (2021). Eliciting auxiliary information for cold start user recommendation: A survey. *Applied Sciences*, *11*(20), 9608.

Adomavicius, G., Jannach, D., Leitner, S., & Zhang, J. (2021). Understanding longitudinal dynamics of recommender systems with agent-based modeling and simulation. *CoRR*, *abs/2108.11068*. https://arxiv.org/abs/2108.11068

Afsar, M. M., Crump, T., & Far, B. (2022). Reinforcement learning based recommender systems: A survey. *ACM Comput. Surv.*, *55*(7). https://doi.org/10.1145/3543846

Amatriain, X., Lathia, N., Pujol, J. M., Kwak, H., & Oliver, N. (2009). The wisdom of the few: A collaborative filtering approach based on expert opinions from the web. *Proc SIGIR*, 532–539. https://doi.org/10.1145/1571941.1572033

Anelli, V. W., Malitesta, D., Pomo, C., Bellogin, A., Di Sciascio, E., & Di Noia, T. (2023). Challenging the myth of graph collaborative filtering: A reasoned and reproducibility-driven analysis. *Proceedings of the 17th ACM Conference on Recommender Systems*, 350–361. https://doi.org/10.1145/3604915.3609489

Antaris, S., & Rafailidis, D. (2021). Sequence adaptation via reinforcement learning in recommender systems. *Proceedings of the 15th ACM Conference on Recommender Systems*, 714–718. https://doi.org/10.1145/3460231.3478864

Armstrong, T. G., Moffat, A., Webber, W., & Zobel, J. (2009). Improvements that don't add up: Ad-hoc retrieval results since 1998. *CIKM '09*, 601–610.

Basu, C., Hirsh, H., Cohen, W., et al. (1998). Recommendation as classification: Using social and content-based information in recommendation. *Aaai/iaai*, 714–720.

Belinkov, Y. (2018). *On internal language representations in deep learning: An analysis of machine translation and speech recognition* [Doctoral dissertation, Massachusetts Institute of Technology].

Belinkov, Y. (2022). Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, *48*(1), 207–219.

Bennett, J., & Lanning, S. (2007). The netflix prize.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Cai, D., Qian, S., Fang, Q., Hu, J., & Xu, C. (2023). User cold-start recommendation via inductive heterogeneous graph neural network. *ACM Transactions on Information Systems*, *41*(3), 1–27.

Canamares, R., Redondo, M., & Castells, P. (2019). Multi-armed recommender system bandit ensembles. *Proc RecSys*. https://doi.org/10.1145/3298689.3346984

Cavenaghi, E., Sottocornola, G., Stella, F., & Zanker, M. (2023). A systematic study on reproducibility of reinforcement learning in recommendation systems. *ACM Transactions on Recommender Systems*, *1*(3), 1–23.

Checco, A., Bianchi, G., & Leith, D. J. (2017). Blc: Private matrix factorization recommenders via automatic group learning. *ACM Trans. Priv. Secur.*, *20*(2). https://doi.org/10.1145/3041760

Chen, C. C., Lai, P.-L., & Chen, C.-Y. (2023). Coldgan: An effective cold-start recommendation system for new users based on generative adversarial networks. *Applied Intelligence*, *53*(7), 8302–8317.

Chen, J., Dong, H., Qiu, Y., He, X., Xin, X., Chen, L., Lin, G., & Yang, K. (2021). Autodebias: Learning to debias for recommendation. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 21–30. https://doi.org/10.1145/3404835.3462919

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 15084–15097, Vol. 34). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf

Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., & Chi, E. H. (2019). Top-k off-policy correction for a reinforce recommender system. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 456–464.

Chen, R., Hua, Q., Chang, Y., Wang, B., Zhang, L., & Kong, X. (2018). A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. *IEEE Access*, *6*, 64301–64320.

Chen, X., Yao, L., McAuley, J., Zhou, G., & Wang, X. (2023). Deep reinforcement learning in recommender systems: A survey and new perspectives. *Knowledge-Based Systems*, *264*, 110335.

Chen, Z., Gan, W., Wu, J., Hu, K., & Lin, H. (2024). Data scarcity in recommendation systems: A survey. *ACM Transactions on Recommender Systems*.

Choi, Y. S. (2014). Content type based adaptation in collaborative recommendation. *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, 61â65. https://doi.org/10.1145/2663761.2666034

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *International conference on computers and games*, 72–83.

Cremonesi, P., & Jannach, D. (2021). Progress in recommender systems research: Crisis? what crisis? *AI Magazine*, *42*(3), 43–54.

Crippa, M., Lanzi, P. L., & Marocchi, F. (2022). An analysis of single-player monte carlo tree search performance in sokoban. *Expert Systems with Applications*, *192*, 116224.

Dalvi, N., Kumar, R., & Pang, B. (2013). Para'normal'activity: On the distribution of average ratings. *Proceedings of the International AAAI Conference on Web and Social Media*, *7*(1), 110–119.

Deffayet, R., Thonet, T., Renders, J.-M., & de Rijke, M. (2023). Offline evaluation for reinforcement learning-based recommendation: A critical issue and some alternatives. *SIGIR Forum*, *56*(2). https://doi.org/10.1145/3582900.3582905

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 4171–4186.

Elahi, M., Braunhofer, M., Gurbanov, T., & Ricci, F. (2018). User preference elicitation, rating sparsity and cold start. *Collaborative Recommendations*, 253–294. https://doi.org/10.1142/9789813275355_0008

Elahi, M., Ricci, F., & Rubens, N. (2016). A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, *20*, 29–50. https://doi.org/10.1016/j.cosrev.2016.05.002

Felício, C. Z., Paixão, K. V., Barcelos, C. A., & Preux, P. (2017). A multi-armed bandit model selection for cold-start user recommendation. *Proc UMAP*, 32â40. https://doi.org/10.1145/3079628.3079681

Feng, J., Xia, Z., Feng, X., & Peng, J. (2021). Rbpr: A hybrid model for the new user cold start problem in recommender systems. *Knowledge-Based Systems*, *214*, 106732.

Ferrari Dacrema, M., Boglio, S., Cremonesi, P., & Jannach, D. (2021). A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)*, *39*(2), 1–49.

Ferrari Dacrema, M., Cremonesi, P., & Jannach, D. (2019). Are we really making much progress? a worrying analysis of recent neural recommendation approaches. *Proceedings of the 2019 ACM Conference on Recommender Systems (RecSys 2019)*.

Ferrari Dacrema, M., Parroni, F., Cremonesi, P., & Jannach, D. (2020). Critically examining the claimed value of convolutions over user-item embedding maps for recommender systems. *Proceedings of the 2020 International Conference on Information and Knowledge Management (CIKM '20)*.

Gao, C., Xu, K., Zhou, K., Li, L., Wang, X., Yuan, B., & Zhao, P. (2022). Value penalized q-learning for recommender systems. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008–2012. https://doi.org/10.1145/3477495.3531796

Golbandi, N., Koren, Y., & Lempel, R. (2011). Adaptive bootstrapping of recommender systems using decision trees. *Proc WSDM*, 595â604. https://doi.org/10.1145/1935826.1935910

Gomez-Uribe, C. A., & Hunt, N. (2015). The Netflix recommender system: Algorithms, business value, and innovation. *Transactions on Management Information Systems*, *6*(4), 13:1–13:19.

Hao, B., Zhang, J., Yin, H., Li, C., & Chen, H. (2021). Pre-training graph neural networks for cold-start users and items representation. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 265–273.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. *Proceedings of the 26th international conference on world wide web*, 173–182.

Hernández-Lobato, J. M., Houlsby, N., & Ghahramani, Z. (2014). Probabilistic matrix factorization with non-random missing data. *International Conference on Machine Learning*, 1512–1520.

Hidasi, B., & Czapp, Á. T. (2023). The effect of third party implementations on reproducibility. *Proceedings of the 17th ACM Conference on Recommender Systems*, 272–282.

Hidasi, B., & Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 843–852.

Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2016). Session-based recommendations with recurrent neural networks. *Proc. ICLR '16*.

Huang, J., Oosterhuis, H., Cetinkaya, B., Rood, T., & de Rijke, M. (2022). State encoders in reinforcement learning for recommendation: A reproducibility study. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2738–2748.

Ie, E., Hsu, C.-w., Mladenov, M., Jain, V., Narvekar, S., Wang, J., Wu, R., & Boutilier, C. (2019). Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*.

Jannach, D., & Jugovac, M. (2019). Measuring the business value of recommender systems. *ACM Trans. Manage. Inf. Syst.*, *10*(4). https://doi.org/10.1145/3370082

Janner, M., Li, Q., & Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 1273–1286, Vol. 34). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2021/file/099fe6b0b444c23836c4a5d07346082b-Paper.pdf

Kaminskas, M., & Bridge, D. (2016). Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems*, *7*(1), 2:1–2:42.

Kang, W.-C., & McAuley, J. (2018). Self-attentive sequential recommendation. *2018 IEEE International Conference on Data Mining (ICDM)*, 197–206.

Klimashevskaia, A., Jannach, D., Elahi, M., & Trattner, C. (2024). A survey on popularity bias in recommender systems. *User Modeling and User-Adapted Interaction*, *34*, 1777–1834.

Labarca, Á., Parra, D., & Icarte, R. T. (2024). On the unexpected effectiveness of reinforcement learning for sequential recommendation. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, & F. Berkenkamp (Eds.), *Proceedings of the 41st international conference on machine learning* (pp. 45432–45450, Vol. 235).

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, C., Yang, Z., Zhang, J., Wu, J., Wang, D., He, X., & Wang, X. (2023). Model-enhanced contrastive reinforcement learning for sequential recommendation. *CoRR*, *abs/2310.16566*. https://doi.org/10.48550/ARXIV.2310.16566

Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., & Ma, J. (2017). Neural attentive session-based recommendation. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 1419–1428.

Li, Y., Liu, K., Satapathy, R., Wang, S., & Cambria, E. (2024). Recent developments in recommender systems: A survey. *IEEE Computational Intelligence Magazine*, *19*(2), 78–95.

Lin, J. (2019). The neural hype and comparisons against weak baselines. *SIGIR Forum*, *52*(2), 40–51.

Liu, F., Tang, R., Li, X., Zhang, W., Ye, Y., Chen, H., Guo, H., & Zhang, Y. (2018). Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027*.

Liu, J., Yang, Z., Li, T., Wu, D., & Wang, R. (2022). Spr: Similarity pairwise ranking for personalized recommendation. *Knowledge-Based Systems*, *239*, 107828. https://doi.org/10.1016/j.knosys.2021.107828

Liu, Q., Zeng, Y., Mokhosi, R., & Zhang, H. (2018). Stamp: Short-term attention/memory priority model for session-based recommendation. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1831–1839.

Lops, P., De Gemmis, M., & Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, 73–105.

Ludewig, M., Latifi, S., Mauro, N., & Jannach, D. (2021). Empirical analysis of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, *31*(1), 149–181.

Ludewig, M., Mauro, N., Latifi, S., & Jannach, D. (2019). Performance comparison of neural and non-neural approaches to session-based recommendation. *Proceedings of the 13th ACM Conference on Recommender Systems*, 462–466.

Marcuzzo, M., Zangari, A., Albarelli, A., & Gasparetto, A. (2022). Recommendation systems: An insight into current development and future research challenges. *IEEE Access*, *10*, 86578–86623.

Milogradskii, A., Lashinin, O., P, A., Ananyeva, M., & Kolesnikov, S. (2024). Revisiting bpr: A replicability study of a common recommender system baseline. *Proceedings of the 18th ACM Conference on Recommender Systems*, 267–277. https://doi.org/10.1145/3640457.3688073

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. *Advances in neural information processing systems*, *20*.

Pan, W., & Chen, L. (2013). Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2691?2697.

Putterman, A. L., Lu, K., Mordatch, I., & Abbeel, P. (2021). Pretraining for language conditioned imitation with transformers.

Quadrana, M., Cremonesi, P., & Jannach, D. (2018). Sequence-aware recommender systems. *ACM Computing Surveys*, *51*.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., & Riedl, J. (2002). Getting to know you: Learning new user preferences in recommender systems. *Proceedings of the 7th International Conference on Intelligent User Interfaces*, 127â134. https://doi.org/10.1145/502716.502737

Rashid, A. M., Karypis, G., & Riedl, J. (2008). Learning preferences of new users in recommender systems: An information theoretic approach. *SIGKDD Explor. Newsl.*, *10*(2), 90â100. https://doi.org/10.1145/1540276.1540302

Ren, Z., Huang, N., Wang, Y., Ren, P., Ma, J., Lei, J., Shi, X., Luo, H., Jose, J., & Xin, X. (2023). Contrastive state augmentations for reinforcement learning-based recommender systems. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 922–931. https://doi.org/10.1145/3539618.3591656

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 452?461.

Rendle, S., Krichene, W., Zhang, L., & Anderson, J. R. (2020). Neural collaborative filtering vs. matrix factorization revisited. *RecSys 2020: Fourteenth ACM Conference on Recommender Systems*, 240–248. https://doi.org/10.1145/3383313.3412488

Rendle, S., Krichene, W., Zhang, L., & Koren, Y. (2022). Revisiting the performance of ials on item recommendation benchmarks. *RecSys '22: Sixteenth ACM Conference on Recommender Systems*, 427–435. https://doi.org/10.1145/3523227.3548486

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*(3), 56–58.

Rohde, D., Bonner, S., Dunlop, T., Vasile, F., & Karatzoglou, A. (2018). Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *CoRR*, *abs/1808.00720*.

Schadd, M. P., Winands, M. H., Herik, H., Chaslot, G. M.-B., & Uiterwijk, J. W. (2008). Single-player monte-carlo tree search. *International Conference on Computers and Games*, 1–12.

Schadd, M. P., Winands, M. H., Tak, M. J., & Uiterwijk, J. W. (2012). Single-player monte-carlo tree search for samegame. *Knowledge-Based Systems*, *34*, 3–11.

Shams, S., Anderson, D., & Leith, D. (2021). Cluster-based bandits: Fast cold-start for recommender system new users. *Proc SIGIR*.

Shehzad, F., & Jannach, D. (2023). Everyone's a winner! on hyperparameter tuning of recommendation models. *17th ACM Conference on Recommender Systems*.

Shi, L., Zhao, W. X., & Shen, Y.-D. (2017). Local representative-based matrix factorization for cold-start recommendation. *ACM Trans. Inf. Syst.*, *36*(2). https://doi.org/10.1145/3108148

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016).

Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Sindhwani, V., Bucak, S. S., Hu, J., & Mojsilovic, A. (2010). One-class matrix completion with low-density factorizations. *2010 IEEE international conference on data mining*, 1055–1060.

Stamenkovic, D., Karatzoglou, A., Arapakis, I., Xin, X., & Katevas, K. (2022). Choosing the best of both worlds: Diverse and novel recommendations through multi-objective reinforcement learning. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 957–965. https://doi.org/10.1145/3488560.3498471

Su, X., & Khoshgoftaar, T. (2009). A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, *2009*. https://doi.org/10.1155/2009/421425

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019). Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 1441–1450. https://doi.org/10.1145/3357384.3357895

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tang, J., & Wang, K. (2018). Personalized top-n sequential recommendation via convolutional sequence embedding. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 565–573.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Volkovs, M., Yu, G., & Poutanen, T. (2017). Dropoutnet: Addressing cold start in recommender systems. *Advances in neural information processing systems*, *30*.

Wan, M., & McAuley, J. (2018). Item recommendation on monotonic behavior chains. *Proceedings of the 12th ACM conference on recommender systems*, 86–94.

Wang, S., Cao, L., Wang, Y., Sheng, Q. Z., Orgun, M. A., & Lian, D. (2021). A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, *54*(7), 1–38.

Wang, S., Chen, X., Jannach, D., & Yao, L. (2023). Causal decision transformer for recommender systems via offline reinforcement learning. *Proceedings of the 46th*

International ACM SIGIR Conference on Research and Development in Information Retrieval, 1599–1608. https://doi.org/10.1145/3539618.3591648

Wang, X., Zhang, R., Sun, Y., & Qi, J. (2019, September). Doubly robust joint learning for recommendation on data missing not at random. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 6638–6647, Vol. 97). PMLR. https://proceedings.mlr.press/v97/wang19n.html

Xin, X., Karatzoglou, A., Arapakis, I., & Jose, J. M. (2020). Self-supervised reinforcement learning for recommender systems. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 931–940.

Xin, X., Karatzoglou, A., Arapakis, I., & Jose, J. M. (2022). Supervised advantage actor-critic for recommender systems. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 1186–1196.

Xin, X., Pimentel, T., Karatzoglou, A., Ren, P., Christakopoulou, K., & Ren, Z. (2022). Rethinking reinforcement learning for recommendation: A prompt perspective. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1347–1357.

Xu, M., Shen, Y., Zhang, S., Lu, Y., Zhao, D., Tenenbaum, J., & Gan, C. (2022). Prompting decision transformer for few-shot policy generalization. *international conference on machine learning*, 24631–24645.

Yamagata, T., Khalil, A., & Santos-Rodriguez, R. (2023). Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. *International Conference on Machine Learning*, 38989–39007.

Yuan, F., Karatzoglou, A., Arapakis, I., Jose, J. M., & He, X. (2019). A simple convolutional generative network for next item recommendation. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 582–590.

Yuan, H., & Hernandez, A. A. (2023). User cold start problem in recommendation systems: A systematic review. *IEEE access*, *11*, 136958–136977.

Zhang, J., Ma, C., Zhong, C., Zhao, P., & Mu, X. (2022). Combining feature importance and neighbor node interactions for cold start recommendation. *Engineering Applications of Artificial Intelligence*, *112*, 104864.

Zhao, K., Liu, S., Cai, Q., Zhao, X., Liu, Z., Zheng, D., Jiang, P., & Gai, K. (2023). Kuaisim: A comprehensive simulator for recommender systems. *Advances in Neural Information Processing Systems*, *36*, 44880–44897.

Zhao, K., Zou, L., Zhao, X., Wang, M., & Yin, D. (2023). User retention-oriented recommendation with decision transformer. *Proceedings of the ACM Web Conference 2023*, 1141–1149.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., & Tang, J. (2018). Deep reinforcement learning for page-wise recommendations. *Proceedings of the 12th ACM Conference on Recommender Systems*, 95–103. https://doi.org/10.1145/3240323.3240374

Zhou, K., Yang, S.-H., & Zha, H. (2011). Functional matrix factorizations for cold-start recommendation. *Proc SIGIR*, 315â324. https://doi.org/10.1145/2009916.2009961