SlateQ for Slate-Based E-Commerce Recommender Systems

Literature and Technology Survey

Muhammad Rasyid Gatra Wijaya

Master of Science in Computer Science
The University of Bath
2024/25

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Traditional recommendation models often focus on suggesting a single item at a time. One common strategy is short-term optimisation, such as fixed greedy policies based on immediate feedback. While effective in the short term, these approaches can fail to capture user preferences that develop over time, which may limit long-term engagement (Zhao et al., 2018). In contrast, most modern systems such as e-commerce platforms, social media, or streaming services recommend a group of items at once, commonly referred to as a slate. This shift introduces a more complex action space, as the number of possible item combinations grows exponentially.

The limitations of short-term optimisation and the complexity of slate-based recommendation challenges have motivated recent research into reinforcement learning approaches that aim to optimise long-term user outcomes (Chen et al., 2023) and address the combinatorial complexity of slate-based recommender systems (Ie et al., 2019b). Reinforcement learning (RL) has therefore been proposed as a promising framework for slate-based recommendation. Unlike contextual bandits, which treat each interaction independently, full RL agents can learn sequential decision policies that account for user history and adapt based on long-term outcomes (Li et al., 2010a). This enables systems to move beyond short-term optimisation and better capture dynamic user preferences over time (Dulac-Arnold, Mankowitz and Hester, 2019). However, applying RL to recommendation tasks introduces significant scalability challenges. Deep Q-networks (DQNs), for example, have demonstrated strong performance in high-dimensional control tasks such as Atari gameplay (Mnih et al., 2015a), but they are not suited for slate recommendation problems, where the number of possible item combinations is extraordinarily large (Ie et al., 2019b).

The SlateQ algorithm (Ie et al., 2019b) addresses this issue by decomposing the slate-level Q-value into per-item components, assuming user choice is conditionally independent across items. This enables efficient Q-learning in environments with large action spaces. Although SlateQ was initially evaluated in RecSim (Ie et al., 2019a), there has been little independent replication or comparison with standard baselines such as DQN or contextual bandits. Furthermore, few studies have explored improvements to SlateQ, such as diversity-aware rewards or enhanced user models. This project addresses these gaps by extending SlateQ with additional techniques and benchmarking it against other RL methods with emphasis on long-term user engagements, specifically in e-commerce settings.

1.2 Aims and Objectives

This research aims to:

- Evaluate the effectiveness of SlateQ in slate-based recommendation environments, focusing on long-term user engagement and behavioural adaptation.
- Benchmark reinforcement learning approaches, including SlateQ, against traditional baselines to understand their advantages in capturing sequential user preferences.

The objectives are to:

- Implement and train a SlateQ agent in RecSim within a simulated e-commerce recommendation scenario.
- Benchmark SlateQ's performance against DQN, contextual bandits, tabular Q-learning, and, if feasible, PPO and IQN.
- Measure sample efficiency, reward progression, and robustness across multiple runs.
- Evaluate recommendation quality using metrics such as NDCG@k, SlateMRR, and cumulative reward.
- Analyse the strengths and limitations of each algorithm in different recommendation scenarios.
- Investigate tuning methods and potential extensions, including diversity-aware rewards, to enhance the effectiveness of SlateQ.
- Assess the practical feasibility of deploying SlateQ in real-world recommendation systems.
- Identify promising directions for future research, including scaling SlateQ to large-scale environments or more complex user behaviours.

Chapter 2

Literature and Technology Survey

2.1 Traditional Recommender Systems

Recommender systems aim to predict user preferences and suggest items that users will appreciate. Early approaches can be broadly categorised into content-based methods and collaborative filtering. Content-based recommenders match users with items sharing similar attributes to those the user has previously liked, using item features and user profiles. In contrast, collaborative filtering (CF) leverages patterns of user-item interactions—users are recommended items that similar users have enjoyed, or items similar to those they previously liked. Memory-based CF algorithms, such as user-based and item-based neighbourhood methods, were among the first successful techniques (Adomavicius and Tuzhilin, 2005). Model-based methods later gained popularity, particularly matrix factorisation, which represents users and items in a latent factor space. The Netflix Prize showcased the effectiveness of matrix factorisation in capturing complex user-item relationships (Koren, Bell and Volinsky, 2009). These traditional methods typically generate a ranked list of individual item recommendations by predicting a relevance score (e.g., a rating or click-through probability) for each item and sorting accordingly.

Over time, more sophisticated models have emerged, including machine learning and deep learning approaches. For instance, factorisation machines and deep neural networks can model complex relationships between features, and recurrent neural networks have been used to handle recommendations that depend on the order of previous user actions, such as in session-based systems. A comprehensive survey by Zhang et al. (2019) covers the evolution of deep learning in recommender systems. Despite these advances, traditional recommenders commonly focus on ranking individual items based on their predicted usefulness. The system usually treats each recommendation opportunity independently, selecting the top-N items for a user based on immediate predicted interest. While effective for one-off recommendations, this one-item-at-a-time strategy has limitations. In particular, it does not naturally account for how recommended items might complement each other as a group, nor how recommending an item now might influence a user's future preferences. These shortcomings have led researchers to explore new paradigms that consider sets of recommendations and sequential user interaction.

2.2 The Shift to Slate-Based Recommendation

In real-world applications like e-commerce websites, social media feeds, or streaming services, users are typically presented with a slate of multiple recommendations at once, rather than a single item. For example, an online storefront might show a page of product suggestions, and a video streaming platform might display a grid of video thumbnails. This shift from individual recommendations to slate-based recommendations reflects practical interface design – multiple options increase the chance a user finds something of interest – but it also introduces new complexity. A slate is more than just a list of separate item suggestions; the items can affect each other in how the user views and picks between them. For example, showing too many similar items might feel repetitive, while a more varied slate could better match different parts of the user's interest. Thus, the problem of selecting an optimal slate goes beyond picking top items individually. It involves diversity, complementarity, and ordering considerations that were less relevant in traditional one-item recommendation settings.

The move to slate-based recommendation also greatly expands the action space for a recommender system. Instead of choosing one item from a catalogue of N items, the system must choose a combination of k items. The number of possible slates grows combinatorially with N and k. For instance, if $N = 20$ and the system needs to display $k = 5$ items, there are approximately:

$$P(20, 5) = \frac{20!}{(20-5)!} = 20 \times 19 \times 18 \times 17 \times 16 = 1{,}860{,}480 \approx 1.86 \times 10^6$$

possible ordered slates—a huge decision space to consider. Even with much larger item catalogues, choosing just a few items for a slate results in an extremely large number of possible combinations. This makes exhaustive search of all slates infeasible.

Because of these challenges, researchers and companies began exploring algorithms designed to work directly with slates. Some approaches, like page-wise recommendation, were introduced to select a group of items together rather than one at a time (Zhao et al., 2018). By treating the whole slate as a single decision, these methods can take into account how the items work together and aim for broader goals—like making the user happy with the whole page or showing a good mix of content. This shift made the problem more complex, but also created space for using stronger decision-making approaches, such as reinforcement learning, in recommender systems.

2.3 Challenges in Slate Recommendation

Using a slate-based approach brings some important challenges that don't come up in traditional recommendation systems:

- **Combinatorial Action Space:** As noted, the space of possible slates is enormous when the item catalogue is large and each slate contains multiple items. This combinatorial explosion makes it intractable to evaluate every possible slate. Effective algorithms must intelligently search or approximate the best slate without brute-forcing all combinations. Traditional greedy strategies (picking top items independently) risk selecting redundant items and miss out on the best overall slate. But trying out every possible combination takes too much computing power (Ie et al., 2019b). This makes it hard to both learn and deliver good slates in practice.

- **Long-Term User Dynamics:** Slate recommendations often happen as part of an ongoing interaction, like a full user session or daily visits over time. If a system only focuses on getting quick clicks, it might miss the bigger picture. For example, showing the same kind of items might work once, but can become boring later (Zhao et al., 2018). The real challenge is to recommend slates that keep users interested in the long run—helping with things like repeat visits or overall satisfaction. To do this, the system needs to think about how today's recommendations might affect what the user wants tomorrow. Most traditional recommenders don't consider this, which is why more advanced, step-by-step decision-making models are needed (Dulac-Arnold, Mankowitz and Hester, 2019).
- **Efficiency and Scalability:** Any algorithm used for slate recommendation has to be fast and efficient, since real systems often need to serve millions of users at once. Algorithms need to prune the search space of item combinations and possibly leverage parallelism or approximations (Ie et al., 2019b). Training these models on large datasets or through simulations can take a lot of computing power. Making sure a slate recommendation system works at industry scale—where there might be millions of items and constant updates—is not easy. To manage this, many methods break the problem into smaller parts or use step-by-step structures to keep things manageable (Dulac-Arnold, Mankowitz and Hester, 2019).

These challenges have motivated researchers to seek new techniques that go beyond static recommendation heuristics. In particular, reinforcement learning (RL) has gained attention as a framework well-suited to handle sequential decision-making and the exploration of a large action space under uncertainty. However, applying RL to slate recommendation isn't straightforward, as it must cope with the combinatorial (permutational when order matters) number of possible slates and learn how users interact with them. Before discussing specialised algorithms like SlateQ that tackle these issues, we first review how reinforcement learning has been used in recommender systems in general.

2.4 Reinforcement Learning in Recommender Systems

Reinforcement learning (RL) treats recommendation as a process that happens step by step. At each interaction, the system suggests one or more items based on what it currently knows about the user—this could include their profile, browsing history, or current situation. After making a recommendation, the system receives feedback from the user, such as a click or a purchase, which it uses to learn and update its understanding of the user's interests. This kind of setup aligns well with the Markov Decision Process (MDP), where the aim is to find a strategy that earns the most total reward over time, instead of just focusing on each recommendation separately.

Early uses of reinforcement learning in recommendation were held back by limited computing power. Shani, Heckerman and Brafman (2005) introduced MDP-based recommenders that track how users move between different states and aim to improve future rewards. However, these early systems usually worked only in simple settings, with a small number of choices and short time spans. A big improvement came with contextual bandits—basic versions of RL that pick one item at a time, based only on the current situation, without planning for the future. Li et al. (2010a) showed that this approach could work at scale by deploying a contextual bandit on Yahoo!'s Front Page using the LinUCB algorithm, which finds a balance between

recommending familiar content and exploring new options. While effective, contextual bandits are not ideal for slate recommendation, since they usually recommend just one item at a time and don't account for how today's choice might affect what the user wants tomorrow.

More recently, deep reinforcement learning (DRL) has been used to deal with more complex user information and large sets of possible recommendations. DRL uses neural networks to help the system learn which actions (like which items to show) are best in different situations. Methods like Deep Q-Networks (DQN) and policy gradient approaches have shown good results in tasks where the system needs to make recommendations over many steps. Dulac-Arnold, Mankowitz and Hester (2019) point out that there are still challenges—like not knowing everything about the user or learning from past data—but there have been real successes. For example, Ie et al. (2019b) mention how a DQN-like method was used in YouTube's recommendation system, showing that RL can work even at a very large scale.

2.5 The SlateQ Algorithm

SlateQ, introduced by Ie et al. (2019b), is an RL algorithm designed to handle the combinatorial nature of slate-based recommendation in a tractable way. In typical recommender environments, users are presented with a *slate*—a set of multiple items—rather than a single recommendation.

SlateQ addresses the exponentially large action space challenge by introducing a decomposition technique that breaks down the slate-level Q-function $Q(s, A)$—the expected cumulative reward from presenting slate A in state s—into a weighted sum of item-level Q-values $Q(s, i)$. This is enabled by two main assumptions: (1) that users select only one item from the slate (Single Choice), and (2) that both reward and state transitions depend only on the consumed item, not the full slate (RTDS: Reward and Transition Dependence on Selection).

Formally, under these assumptions and using a choice model such as the Multinomial Logit, the value of a slate A can be approximated as:

$$Q(s, A) = \sum_{i \in A} P(i \mid s, A) \cdot Q(s, i)$$

where $P(i \mid s, A)$ is the probability of a user selecting item i from slate A in state s. This transforms the combinatorial slate optimisation into a linear programming problem, or even simpler greedy heuristics.

For training, the item-level Q-values are updated using a decomposed temporal difference rule:

$$Q(s, i) \leftarrow \alpha \left(r + \gamma \sum_{j \in A'} P(j \mid s', A') Q(s', j) \right) + (1 - \alpha) Q(s, i)$$

where A' is the next slate and s' the subsequent state. At serving time, the agent constructs a slate by selecting the top-k items with the highest $v(s, i) \cdot Q(s, i)$ scores, where $v(s, i)$ is an unnormalised proxy of the user's interest (e.g., from a click-through-rate model).

This decomposition enables generalisation across slates, improves sample efficiency, and is easily integrated into systems like YouTube, where SlateQ was deployed in a live experiment. The algorithm significantly improved user engagement over baseline myopic recommenders, which demonstrates its scalability.

2.6 Other Reinforcement Learning Algorithms

Beyond SlateQ, a number of reinforcement learning and bandit algorithms have been applied to recommender systems. This project benchmarks SlateQ against several of these baselines to understand their relative strengths. Below is a summary of each:

- **DQN (Deep Q-Network):** DQN is a value-based method that uses neural networks to estimate action values (Mnih et al., 2015b). In recommendation settings, it has been used to recommend one item per step, learning which choices lead to higher long-term reward. However, DQN struggles with slate recommendations due to the exponential growth in the number of item combinations. In this project, DQN serves as a baseline that selects a single item at each step.
- **Policy Gradient Methods (e.g., PPO):** These methods learn a policy directly rather than estimating value functions. PPO (Proximal Policy Optimisation) is widely used due to its stable training performance (Schulman et al., 2017). Zhao et al. (2018) showed that actor-critic methods can output a full slate of items. In theory, PPO can handle large action spaces by sampling, but unlike SlateQ, it lacks structural assumptions, which can make training less efficient in sparse reward settings.
- **Distributional RL (e.g., IQN):** Instead of just learning the average expected reward, distributional reinforcement learning looks at the full range of possible rewards. IQN (Implicit Quantile Networks) (Dabney et al., 2018b) helps the system understand uncertainty in how users might respond, which can lead to better exploration and decision-making.
- **Contextual Bandits:** These algorithms make decisions based on the current user context without modelling long-term impact. LinUCB (Li et al., 2010b) is a well-known method that balances exploration and exploitation. Contextual bandits are commonly deployed in industry for short-term metrics like click-through rate. They serve here as a baseline for comparison with more complex sequential models.
- **Tabular Q-Learning:** We include a basic tabular Q-learning agent in a small, simplified environment to use as a control. While this approach wouldn't work for large or complex systems, it helps us test whether our reinforcement learning setup is working properly in a more controlled setting.
- **Other Approaches:** There are also cascading bandits (Kveton et al., 2015) and newer frameworks like SlateFree (Giovanidis, 2022), which extend SlateQ's decomposition principles. While these methods may not be explored in this project, they highlight the ongoing innovation in slate-based recommendation systems.

2.7 Evaluation Framework: RecSim

Evaluating RL algorithms for recommender systems in real-world settings is both challenging and risky. Live experiments can be slow, expensive, and potentially harmful to user experience if an underperforming policy is deployed. To support experimentation without such risks, researchers often rely on simulation environments. One such tool is RecSim, an open-source simulation platform developed by Google for modelling realistic recommender environments (Ie et al., 2019a).

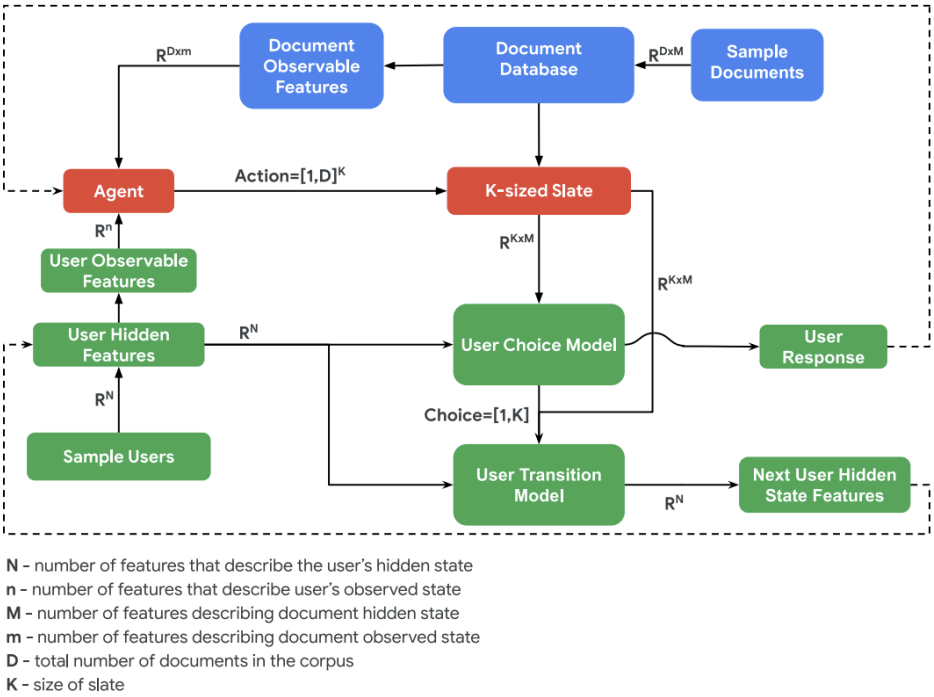

Figure 2.1: RecSim framework for simulating user–item interactions (Ie et al., 2019a).

RecSim is a customisable simulation platform that models a Reinforcement Learning environment, in which a recommender agent interacts with a collection of documents (or recommendable items) and a set of user profiles, to reinforce the algorithm's development. As can be seen in Figure 2.1, the environment is composed of a user model, a document model, and a user choice model. The recommender agent interacts by recommending slates of documents in fixed or dynamic length to users. The agent has visibility over the features of user profiles and candidate documents to create recommendations.

The user model draws users from prior distribution defined over (customisable) user features. These may include latent attributes such as, interests, personality, and satisfaction, observable characteristics such as demographics, and behavioral attributes, such as, session length, visit frequency, or time budget. The document model characterises items from a prior distribution over document attributes, such as document quality, and observable attributes such as topic, document length, and global statistics like ratings and popularity. The level of visibility for user and document attributes is customisable. This design provides flexibility to suit different research objectives (Ie et al., 2019a).

The agent observes user and document features to generate a K-sized slate of documents. The user response to this slate is then determined by a user choice model, based on both the slate and user attributes. After a document is consumed, the user's internal state is updated via a customisable user transition model. For example, a user's interest in a document topic may increase or decrease. Their remaining time budget may decrease at different rates depending on the quality of the document. In addition, the user's satisfaction may vary based on the match between their interests and the document's quality (Ie et al., 2019a).

The main advantage of RecSim is that it treats the recommendation task as a reinforcement learning loop. At each time step, the agent observes the current state, which may include the

user's profile, history, or other context, and selects an action (a slate of items to recommend). The simulated environment then provides a response in the form of a reward and updates the user state accordingly. This setup makes RecSim particularly well suited for studying long-term user engagement and changing preferences, which are critical challenges in domains such as e-commerce and content recommendation.

RecSim enables researchers to build custom simulation setups by defining user behaviour models, item or document attributes, and interaction rules. This allows safe, controlled, and repeatable testing of different recommendation strategies without involving real users. It ensures fairness and consistency in evaluation, and also improves reproducibility. In this project, RecSim serves as the primary evaluation tool. While it does not fully replicate production-scale systems, it provides a useful balance between realism and control. Its ability to simulate slate choice, user satisfaction, and long-term feedback makes it a practical and reliable environment for comparing the performance of RL algorithms.

2.8 Research Gaps and Project Justification

The survey above shows that using reinforcement learning for slate-based recommendation is a growing and promising research area. However, several gaps in research and practice remain, which this project aims to address.

- **Benchmarking:** Although some previous studies have compared individual reinforcement learning algorithms to simpler baselines, there is limited research that evaluates multiple RL methods side by side within a consistent experimental setup. This project aims to address that gap by benchmarking SlateQ against alternative approaches—such as DQN, PPO, IQN, contextual bandits, and tabular Q-learning—using the same RecSim environment. This comparative analysis can help determine which methods perform best under different conditions and whether SlateQ's reported advantages hold up under fair and consistent evaluation.
- **Diversity:** Good slate recommendations often depend not just on immediate clicks but also on providing variety. SlateQ does not explicitly optimise for diversity. This project investigates whether long-term planning in SlateQ leads to naturally diverse slates or whether it concentrates on similar items. Findings may suggest ways to enhance reward functions or algorithm structure in future work.
- **Reproducibility:** By using RecSim and reporting standardised results, this project contributes to the reproducibility of RL research in recommender systems. The experimental setup and findings can serve as a benchmark for future studies evaluating slate-based algorithms.
- **Potential Improvements:** The project also investigates whether techniques such as diversity-aware rewards, enhanced user modelling, or incorporating elements from distributional RL (e.g., IQN) could improve SlateQ's performance. While these extensions may be exploratory, they could offer insights into how SlateQ might be adapted or extended in future work.

2.9 Future Work

While this project focuses on evaluating SlateQ and several baselines within the RecSim environment, the findings may suggest directions for future research. Some potential extensions include:

- **Diversity Objectives:** Most existing algorithms focus on engagement metrics such as clicks or watch time. However, promoting diversity can lead to better user satisfaction and long-term outcomes (Vargas and Castells, 2011).
- **User Modelling Enhancements:** While RecSim supports flexible user modelling, it could be further extended by incorporating delayed feedback, user fatigue, and other behavioural factors (Ie et al., 2019a). These improvements would allow for testing how algorithms perform under more realistic and varied conditions.
- **Hybrid Approaches:** Combining techniques—for example, SlateQ's value decomposition with IQN's ability to capture return uncertainty—may lead to better performance. Prior work suggests hybrid models can adapt more effectively to complex environments (Dabney et al., 2018a).
- **Real-World Deployment Studies:** SlateQ has been tested at YouTube scale (Ie et al., 2019b), but many other algorithms have not. Future studies could focus on real-world deployment challenges such as latency, resource usage, and explainability.

Bibliography

Adomavicius, G. and Tuzhilin, A., 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Ieee trans. knowl. data eng.*, 17(6), pp.734–749.

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

Dabney, W., Ostrovski, G., Silver, D. and Munos, R., 2018a. Implicit quantile networks for distributional reinforcement learning. *International conference on machine learning (icml)*, 80, pp.1096–1105.

Dabney, W., Rowland, M., Bellemare, M.G. and Munos, R., 2018b. Implicit quantile networks for distributional reinforcement learning. *Proceedings of the 35th international conference on machine learning (icml)*. pp.1096–1105.

Dulac-Arnold, G., Mankowitz, D.J. and Hester, T., 2019. Reinforcement learning in recommender systems: A complex answer to a simple problem. *Icml 2019 workshop on real world reinforcement learning*.

Giovanidis, A., 2022. Slatefree: A model-free decomposition for reinforcement learning with slate actions. *arxiv preprint arxiv:2209.01876* [Online]. Available from: `https://arxiv.org/abs/2209.01876`.

Ie, E., Hsu, C.W., Mladenov, M., Jain, V., Narvekar, S., Wang, J., Boutilier, C. et al., 2019a. Recsim: A configurable simulation platform for recommender systems. *arxiv preprint arxiv:1909.04847*.

Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R. et al., 2019b. Slateq: A tractable decomposition for reinforcement learning with recommendation slates. *Proceedings of the 28th international joint conference on artificial intelligence (ijcai)*, pp.2592–2599.

Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8), pp.30–37.

Kveton, B., Szepesvari, C., Wen, Z. and Ashkan, A., 2015. Cascading bandits: Learning to rank in the cascade model [Online]. In: F. Bach and D. Blei, eds. *Proceedings of the 32nd international conference on machine learning*. Lille, France: PMLR, *Proceedings of Machine Learning Research*, vol. 37, pp.767–776. Available from: `https://proceedings.mlr.press/v37/kveton15.html`.

Li, L., Chu, W., Langford, J. and Schapire, R.E., 2010a. A contextual-bandit approach to

personalized news article recommendation. *Proceedings of the 19th international conference on world wide web*. pp.661–670.

Li, L., Chu, W., Langford, J. and Schapire, R.E., 2010b. A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th international conference on world wide web*. pp.661–670.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2015a. Human-level control through deep reinforcement learning. *Nature*, 518(7540), pp.529–533.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015b. Human-level control through deep reinforcement learning. *Nature*, 518, pp.529–533.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arxiv preprint arxiv:1707.06347*.

Shani, G., Heckerman, D. and Brafman, R.I., 2005. An MDP-based recommender system. *Journal of machine learning research* [Online], 6, pp.1265–1295. Available from: `https://www.jmlr.org/papers/v6/shani05a.html`.

Vargas, S. and Castells, P., 2011. Rank and relevance in novelty and diversity metrics for recommender systems [Online]. *Proceedings of the fifth acm conference on recommender systems*. ACM, pp.109–116. Available from: `https://doi.org/10.1145/2043932.2043955`.

Zhang, S., Yao, L., Sun, A. and Tay, Y., 2019. Deep learning based recommender system: A survey and new perspectives. *Acm computing surveys*, 52(1), pp.5:1–5:38.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D. and Tang, J., 2018. Deep reinforcement learning for page-wise recommendations. *Proceedings of the 12th acm conference on recommender systems*. pp.95–103.

Appendix A

Project Plan

A.1 Initial Gantt Chart

The project will run across the dissertation period. For a rough breakdown, see Figure A.1.

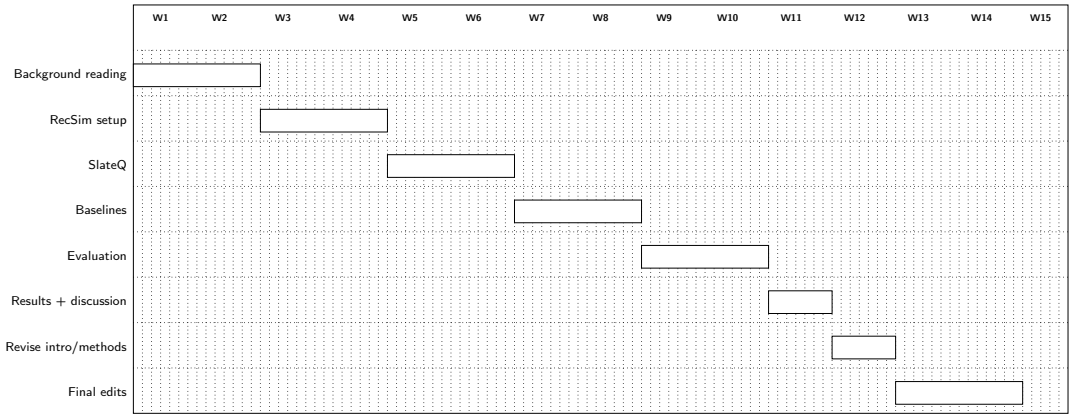

Figure A.1: Gantt chart showing project timeline over 14 weeks.

A.2 Risks and Mitigation

- **Baseline limits**: Some RL agents, such as PPO or IQN, may take a long time to train or may not be successfully implemented. If either of them causes difficulties, one or both may be removed from the project to save time.
- **Layout**: To reduce scope, only one display layout (Grid) will be implemented.
- **Item attributes**: Initially, only the 'category' feature will be used to simplify simulation. Other features (e.g., price or rating) may be added if time allows.

Appendix B

Requirements

B.1 Functional Requirements

- Develop or adapt RL agents: SlateQ (main), DQN, Bandit, IQN, PPO, and Tabular-Q-Learning.
- Use Google RecSim to simulate user-item interactions and collect data.
- Train all agents using consistent settings, evaluation pipelines, and random seeds for reproducibility.
- Tune hyperparameters for all agents and ensure fair comparisons across models.
- Log metrics like average return, click-through rate, diversity, convergence time, and long-term engagement.
- Visualise training curves, agent performance, and convergence behaviour across models.
- Compare SlateQ's performance against traditional baselines (e.g., DQN, Bandit) to evaluate its effectiveness in the recommendation environment.

B.2 Non-Functional Requirements

- Inference for trained models should be fast enough to process interactions quickly in a simulated environment, so the system can run efficiently and simulate real-time recommendations.
- Ensure the system is scalable for handling large datasets and user interactions.
- Clear and thorough documentation of code, configurations, and evaluation methodology for reproducibility and ease of use.
- Make models easy to deploy and integrate with other systems to make them user-friendly for future development.

Appendix C

Resources

C.1 Software

- Python for implementation, scripting, and experimentation.
- TensorFlow (v2.x) to run Google's RecSim environment.
- PyTorch for the algorithms implementations.
- Matplotlib and Seaborn for data visualisation and result analysis.

C.2 Training

- University of Bath's Hex GPU cluster.
- Docker (via Hare), as it is the required container orchestration tool for running GPU workloads in the Hex environment.
- If needed, use of third-party GPU instances, provisioned personally.

C.3 Data

I plan to generate a synthetic dataset that includes:

- **Categories:** I will define several product categories (e.g., electronics, homeware, books, etc.) to model user preferences across various types of products.
- **Items:** These items will have key attributes like product name, price, category, and average rating. I'll generate realistic product data to simulate real-world interactions.
- **User Interactions:** I will simulate user interactions based on preferences derived from the categories, such as clicks or purchases. These interactions will be recorded along with the timestamp and the user's remaining session time.

AI tools will be used to generate realistic item names, descriptions, and ratings to make the dataset compatible with RecSim. The synthetic data will reflect typical user behaviour and item interactions based on the earlier RecSim setup. This allows the SlateQ algorithm to be trained and tested in a controlled environment.