CSCI 566 - Deep Learning and its Applications (Spring 2024)
Course Project - Midterm Report
Reinforcement Learning for Recommender Systems - Environment Suite

By: Pranav Parnerkar

Assigned TA:
Ayush Jain

Summary:
As part of our exploration into the realm of reinforcement learning (RL) applied to recommender systems, we delve into the development of a set of environments tailored for agent experimentation and assessment. This report encapsulates our venture into constructing a Netflix recommendation environment using RecSim, detailing the key milestones we achieved along the way. Beginning with an extensive literature review, we delve into previous research in the field and elucidate the features and advantages of RecSim. Subsequently, we outline our workflow, which encompasses pivotal stages such as data preprocessing, feature engineering, and environment design, all geared towards optimizing Netflix recommendations. The report provides an in-depth overview of our environment's components, including document and user models, response simulation, and reward mechanisms. Furthermore, it outlines our analysis using a random agent as a benchmark and informs us of our plans to explore more sophisticated agents like the Full Slate Q and the Cluster Bandit Agents. Preliminary results shed light on the current operation of our environment and serve as a foundation for refining it as we move forward. Our future endeavours involve benchmarking various agents, expanding our approach to encompass platforms beyond Netflix, and enhancing the suite of environments for wider applicability.

Literature Review:
This review will focus on RecSim, a prominent environment suite used for building simulation environments and benchmarking various Reinforcement Learning (RL) agents in the context of recommender systems. Publications [1] and [2] highlight RecSim, emphasizing its role in training and assessing RL agents in recommender systems by offering realistic simulation environments. RecSim, in conjunction with the RL4RS dataset, enables the modelling of user behaviours, defining actions, and state representations - all essential for understanding contextual elements and user-item interactions. The controlled environment allows comparisons between RL algorithms facilitating systematic analysis of hyperparameters' impact on the agent's performance. Despite its benefits, RecSim-based benchmarks face challenges such as simulating real-world user behaviour and scalability issues, highlighting the importance of cautious interpretation of results & addressing limitations in simulation accuracy.

Papers [3] and [4] explore particular approaches to improve recommender systems. Netflix's case study investigates the integration of deep learning approaches, including bag-of-video and sequential models to maximize customer retention. Time-based information enrichment enhances deep learning's capabilities, though challenges persist in accurately predicting online performance. Similar to how the e-commerce search engine application formalizes the problem, the reinforcement learning approach uses full backups and deterministic policy gradients to estimate value functions with high accuracy. The algorithm tackles issues specific to e-commerce search ranking while giving priority to maximizing long-term cumulative benefits. Using deep learning and reinforcement learning strategies to improve system performance and user experience, these approaches depict the advancement of personalized recommendation systems.

Workflow:

- **Dataset:**
  Our recommendation system for Netflix relies on the "Netflix audience behaviour - UK movies" dataset, capturing user interactions like genre preferences and viewing durations. We've enriched our analysis with data from the IMDB movie dataset, adding a "rating" column for each movie. This allows us to gauge document quality and tailor recommendations based on both user behaviour and perceived content quality, ensuring personalized and engaging suggestions for our users.

- **Feature Engineering and Data Ingestion**
  Often referred to as the core of learning-based endeavours, feature engineering stands as a crucial and perhaps inventive process. It converts raw data into valuable formats suitable for exploratory data analysis, ultimately facilitating model/agent training and validation. Unlike deep learning, wherein this step is more often than not automated, in reinforcement learning, it's crucial that the right features are fed (and in the right manner) to the agent while training. Features in reinforcement learning are used to define state representations, action space definitions, and for reward engineering.

  - **Current feature engineering steps and the thought behind 'engineering' them:**

1. Genre Feature Vector - Each movie or show in the dataset was associated with a list of relevant genres. During data analysis, we observed that not all genres had equal significance. To optimize computational resources, we decided to focus on genres with a frequency of occurrence greater than the median value. This resulted in a genre feature vector consisting of thirteen binary entries, each indicating the presence or absence of a popular genre.
2. Cluster ID: Integer values derived from the binary representation of the genre feature vector. These IDs are utilized by RecSim to sample recommendations.

3. IMDb Rating and Total Runtime: These metrics were compiled from various datasets for the movies and shows in our dataset. IMDb Rating serves as an additional feature for each movie or show, while Total Runtime is used to calculate the 'watchtime fraction,' indicating the percentage of the movie or show watched by the user between clicks.
4. Word2Vec Embedding for Movie Titles: This embedding is utilized as an additional feature for the movie or show titles in the dataset.
5. User Interest Vector: We aggregated the dataset based on a unique user identifier called 'user_id', and then calculated a user interest vector for every user by averaging genre feature vectors of the movies or shows they watched. This aggregated vector serves as the primary user-related feature in RecSim, providing insights into individual user preferences.

- **Data Ingestion:**
  RecSim provides entry points within its samplers, enabling us to connect our features to the environment seamlessly without the need for extensive modifications to the framework.

- **Netflix Environment Design:**
  After making a recommendation, the agent is provided - set of available documents, user's latent state, user's response to last recommendation and the reward. Based on these values, it updates its policy.
  1. Users are characterized by user_interests vectors and rating weights for document ratings.
  2. Documents possess feature vectors and ratings. Relevance scores are computed by taking the dot product of user interests and document features via the score_documents() function.
  3. The agent recommends a slate of size K to users, who respond by clicking on a video. The MultinomialLogitChoiceModel employs softmax to determine the chosen document.
  4. User responses simulate watchtime, updating user interests and time budget through the update_state() function.
  5. Terminal conditions are checked after each state transition. If the session continues, the agent recommends another slate, iterating the process.

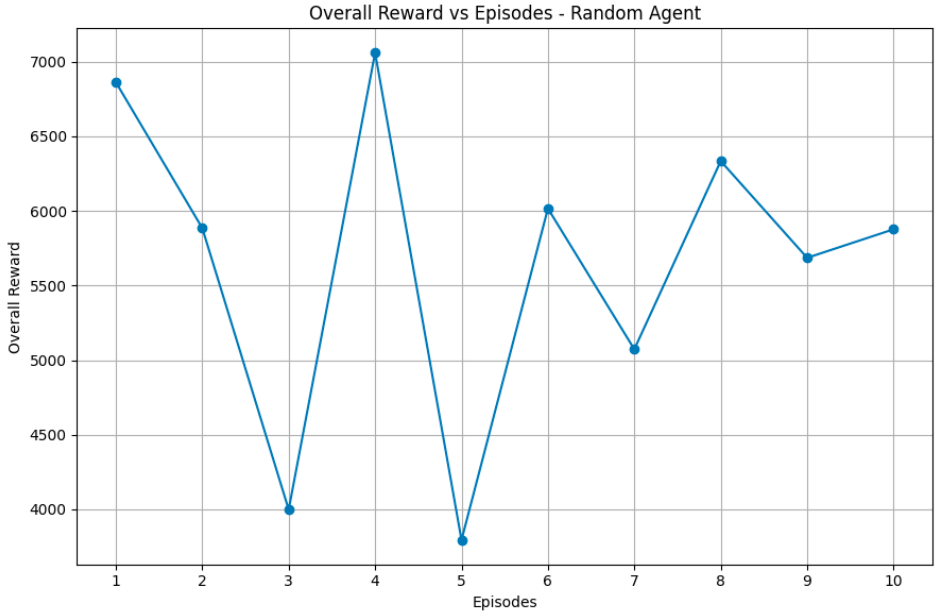
Component	**Attributes**	**Methods**
Document Model (Video Object)	• `Video Feature Vector: [genre1, …actual_quality, actual_length, release_year, director]` • `IMDb Rating` • `Video Watch Time` • `Cluster_ID, Doc_ID`	-
User State	• `User interest /user history: [genre1_weight, .., quality_weight, movie_length_wieght, release_year_weight, director_weight]` • `step_penalty`	**Score Document:** Dot product of User's feature array and Item's feature array. Must reflect actual data - score should be high for User-Video pair that's present in data.
User Model	• `slate_size` • `choice_model_ctor` • `response_model_ctor=NetflixResponse` • `user_state_ctor=NetflixUserState` • `no_click_mass` • `seed` • `alpha_x_intercept` • `alpha_y_intercept`	**Update State:** if clicked on a video: • Update user interest vector: calculate genre mask as given in interest evolution env. Update user interest based on watch time and click. If watch time is low then user-interest will be updated by a small value. Else user interest += mask. • Update time budget (play around): decrease time budget as per watched video time Increase time budget as user is inclined to binge watch else: decrease time budget by a fixed penalty **Is terminal:** Check if time budget is positive
Response Model	• `is_clicked,`	**Simulate Response:**

	• watch_time, • quality, • cluster_id	• Calculate watch time using relevance score(from score_document). Same as calculating engagement scores. Also for the user-video combination given in actual data, the calculated watch time score must be high. That will test our watch time function • Set Click value • Set Quality as IMDb rating • Cluster_ID - Genre **Reward Function:** • A function of clicked and watch time, with more weight to watch time.

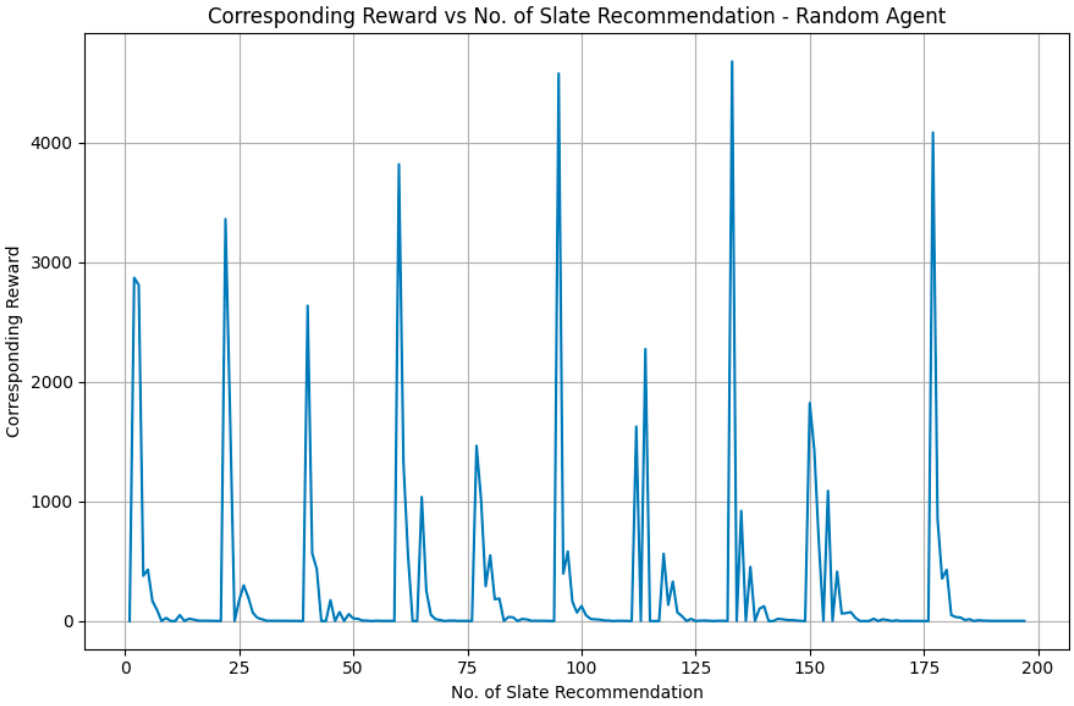
- **Agents:**
  Currently, our investigation has focused on the RandomAgent, where we thoroughly documented our observations and outcomes. Over the next few weeks, we plan to train the existing environment using these agents to further enhance our understanding and evaluation of their performance within the system. Since we have to recommend multiple items at once, we will be exploring Full Slate Q Agent and Cluster Bandit Agent.
  1. ***Random Agent:*** The RandomAgent, a basic recommendation agent provided by RecSim, serves as a benchmark for evaluating the performance of more sophisticated recommendation algorithms. It randomly selects items to recommend without considering user preferences or system dynamics. While simplistic, it provides a baseline against which the effectiveness of more advanced agents can be measured.
  2. ***Full Slate Q Agent:*** It focuses on giving the entire slate(multiple documents) as output. It uses Q Learning & aims for long-term value, i.e to keep users engaged over time. It goes beyond maximizing immediate clicks & considers how a recommended slate might influence user behavior over multiple sessions. Full Slate Q Agent takes observation space & action space which will be accessed once the environment is created. UserState & Document classes have observation_space() which will return observations created by create_observation() in the form of OpenAI Gym's spaces.Box.
  3. ***Cluster Bandit Agent:*** It recommends items with highest UCB of topic affinities. It assumes no knowledge of the user's affinity for each topic. The agent receives observations of the user's past responses for each topic. Bandit Algorithm is utilized to pick the best topics while creating a slate. Documents with the best quality scores are picked within the same best topic.

Results and Analysis using a Random Agent for one of the users:

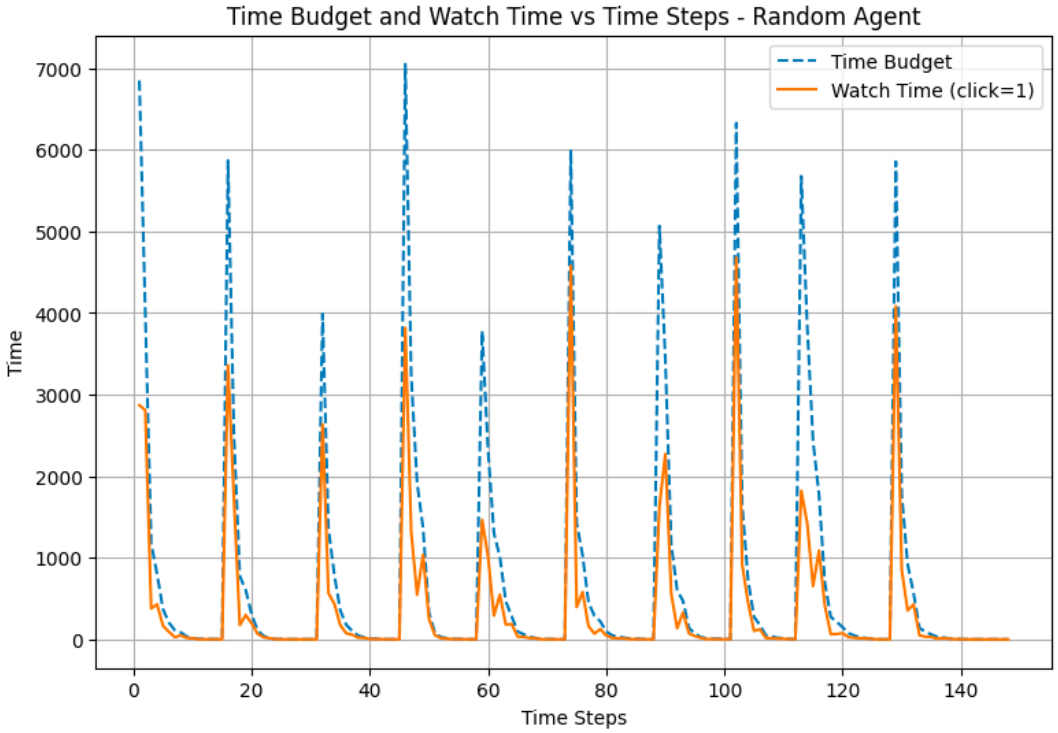
1. **Overall Rewards vs Episodes**: The following plot shows us how the overall reward score changes during the random agent's training. Random agents are generally used as baselines hence we can see that there is virtually no learning here.



2. **Reward vs No. of Slate Recommendation:** The plot below depicts the reward trend across training episodes. Initially in an episode, rewards are notably high due to the simulated user's ample time budget. However, they sharply decrease when the user engages with time-intensive recommendations. With random agents these dips are drastic but in case of policy-based agents, we expect them to be uniform. Abrupt peaks after a consistent zero reward mark the start of a new training episode.

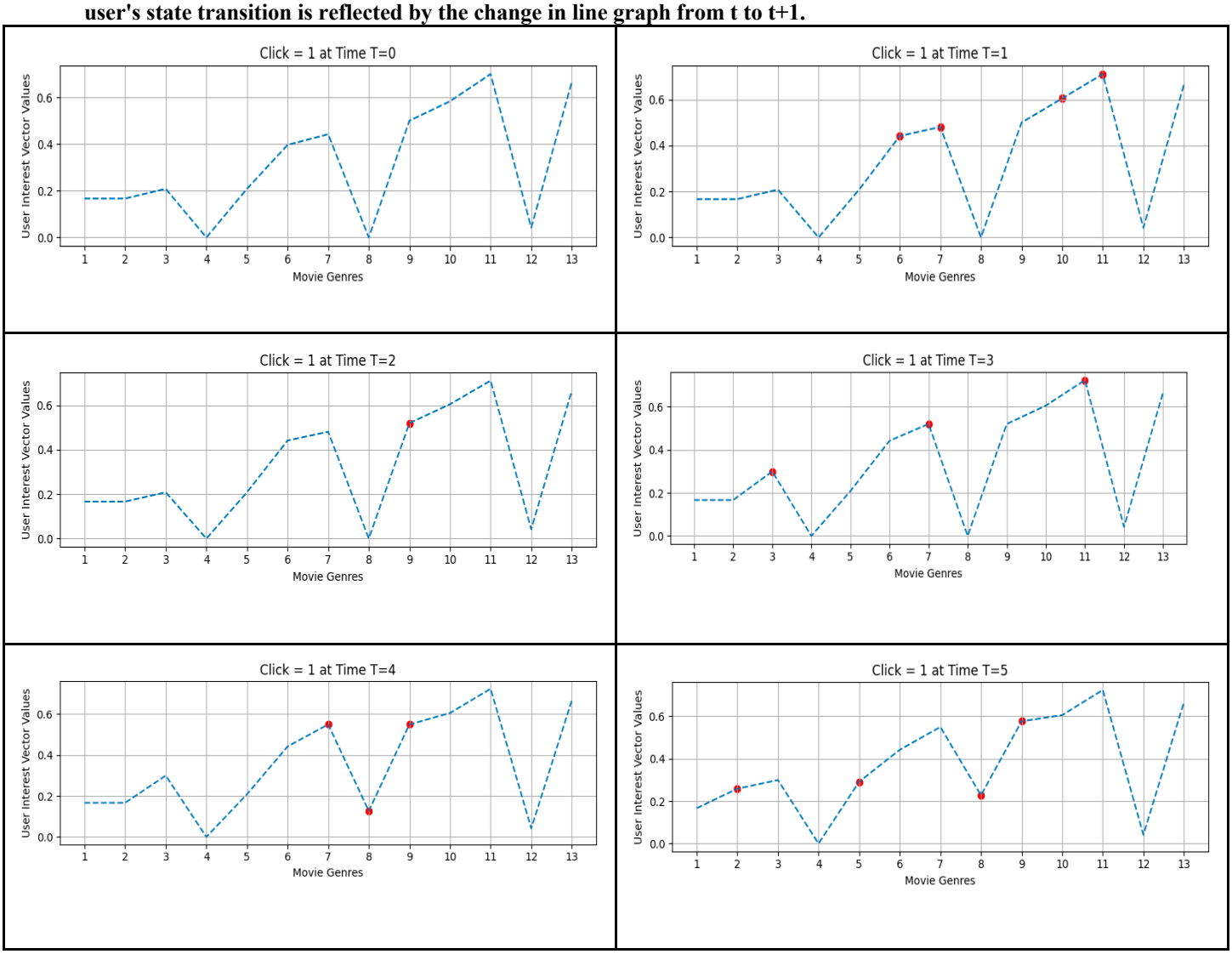


3. **Time Budget and Watch Time vs Time Step:** This visualization illustrates the correlation between the trends of time budget and watch time across consecutive time steps. We generated this plot to verify the correctness of the RecSim environment.

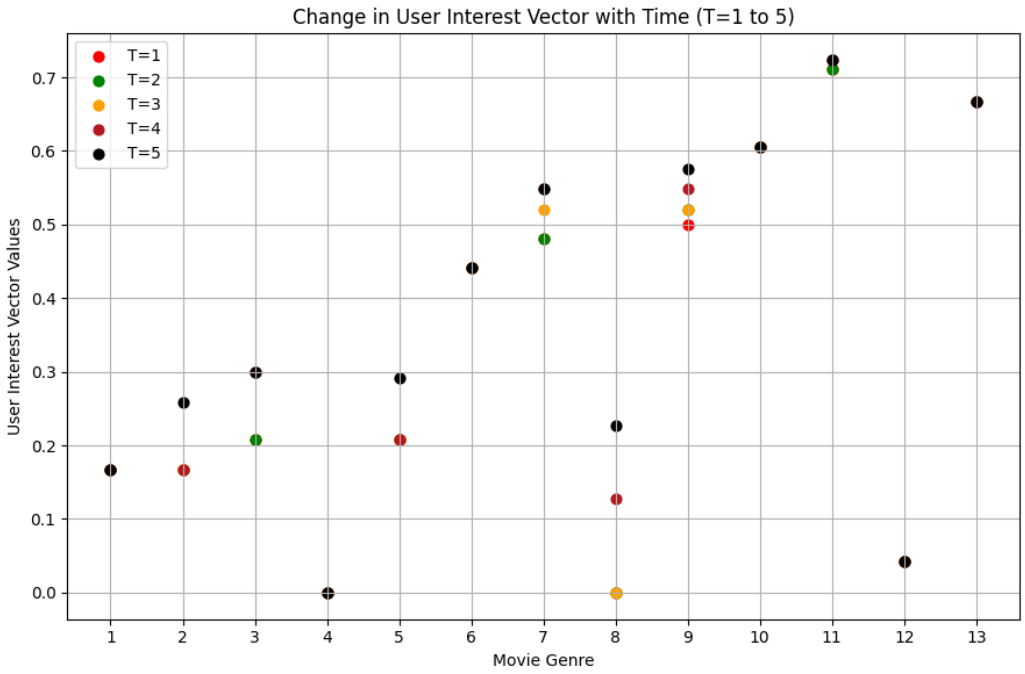


4. **Change in User Interest Vector with Time**: The below graphs display how user interests change due to the interaction with the clicked video. User's interest in a particular genre g_i grows if they click on a video which belongs to genre g_i. Let U_t

be the user's interest vector (preference the user gives to each genre) at time t, and V_t be the video feature vector. U_t is denoted by a blue line and red points indicate V_t (a red point against a genre indicates that genre is set for that V_t).
Note: V_t - which is the video clicked by the user at time t is actually plotted at time t+1. So the effect of V_t on the user's state transition is reflected by the change in line graph from t to t+1.



In the above plots we can observe that the user's interests in each genre grows if the clicked video belongs to that genre.



Overall user's interest vector update is captured in the above plot

Next Steps:

Our immediate objective is to conduct a benchmarking exercise using the diverse range of agents offered by RecSim within our Netflix recommendation system environment. By evaluating the performance of agents such as the Full Slate Q Agent and the Cluster Bandit Agent, we aim to gain valuable insights into their efficacy and suitability for our specific use case. Furthermore, we envision leveraging the knowledge and experience gained from this benchmarking process to extend our efforts beyond Netflix. With the same methodology and approach, we plan to develop recommendation systems tailored for other prominent platforms such as e-commerce, and healthcare industries. Ultimately, our goal is to create a suite of environments encompassing various platforms and industries, thereby maximizing the impact and versatility of our recommender system solutions.

References:

[1] Kai Wang, Zhene Zou, Minghao Zhao, Qilin Deng, Yue Shang, Yile Liang, Runze Wu, Xudong Shen, Tangjie Lyu, Changjie Fan. " RL4RS: A Real-World Dataset for Bridging the Gap in RL-based Recommender Systems" arXiv preprint arXiv:2110.11073 (2021)

[2] Ie, Eugene, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. "Recsim: A configurable simulation platform for recommender systems." arXiv preprint arXiv:1909.04847 (2019)

[3] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, Justin Basilico. "Deep Learning for Recommender Systems: A Netflix Case Study." AI Magazine. doi: 10.1609/aimag.v42i3.18140 (2021)

[4] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, Yinghui Xu. "Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application" arXiv preprint arXiv:1803.00710v3 (2018)