Reinforcement Learning for Recommender Systems - Environment Suite

Pranav Parnerkar
parnerka@usc.edu

Abstract

Recommender Systems have been a part of the digital ecosystem since the inception and rapid proliferation of search engines. Initially, these systems were easy to implement and interpret as they leveraged mathematically formalized algorithms. Mathematically powered rec-systems were eventually superannuated as they lacked multifacetedness by design. With the advent of complex deep-learning architectures in the early 2010s, rec-systems pivoted from being interpretable to convoluted black boxes for capturing better dynamics in big data. However, as we prioritize making AI more robust, researchers are looking to incorporate reinforcement learning (often considered the only way to achieve artificial general intelligence) into rec-systems to make them more reliable, interpretable, and potentially better than its hip deep learning-based variants. This change brings a ton of technical challenges with it. In this project, we aim to tackle one such challenge. Currently, the RL-RecSys community lacks a unified suite of RL-RecSys environments simulating different kinds of RecSys scenarios, like shopping recommendations (Amazon), video recommendations (Netflix, YouTube), educational recommendations (Ed-tech assistants), etc. Our objectives will be to build a RL-RecSys environment for video recommendations (like Netflix and YouTube), benchmark simple RL agents on the developed environment and identify relevant RL-RecSys challenges/future scope in the process.

1 Introduction

Let's consider a scenario - you have an MLE job interview at OpenAI (congratulations on beating their ATS system, by the way), and as we all do, you are scrambling at the last minute. From mindlessly scrolling Reddit/Medium/LinkedIn posts, you've got to know that interviews in the past have been deep learning heavy. YouTube is your virtual guru, and you start binging on deep learning concepts, tutorials, and mock interviews. After a couple of hours, you realize that your recommendations have saturated, and now you are watching repetitive content (though from different creators). Expressing a momentary dissatisfaction with YouTube, you will subsequently initiate searches for specific content in the search box to avoid wasting time. Eureka - this happens because deep learning-based rec-systems often suffer from two problems - myopia and system-induced bias.

Let's dive into them individually:

- **Myopia**: These models try to pigeonhole recommendations that are more likely to lead to an immediate user response (aka click-bait) instead of long-term utility.
- **System-induced bias**: These models try to replicate similar click flows based on a population sample's history, thereby inherently inducing bias to one form/flow of content for overall optimization rather than that specific user's preferences.

These problems are difficult to rectify in the current rec-systems because of their black-box nature. Reinforcement learning-based rec-systems, however, would be significantly more interpretable in nature, and if these two problems arise, then they would be easier to pinpoint and address programmatically. An interpretable system also has the potential to tell us how it made a recommendation based on its underlying mathematical foundation, thereby giving us room to explore and build better mechanisms gradually.

Framing the Recommendation Problem into the Reinforcement Learning Setup: The recommendation problem is a unique instance of a reinforcement learning problem whereby the user is the environment upon which the agent, the recommendation system, acts to receive a reward - a click or engagement by the user [3]. Learning is interpretable, wherein a reward/penalty is provided to the recommendation agent for optimizing the underlying model or policy.

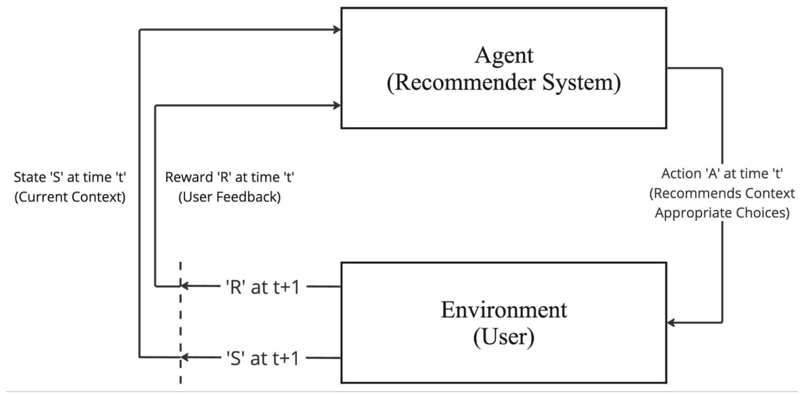


Figure 1: Framing the recommendation problem into the reinforcement learning setup

In conclusion, utilizing reinforcement learning in recommender systems appears to be a strategic approach for addressing the aforementioned challenges, potentially enhancing sustained user engagement with recommendations over time. In this project, we aim to address the lack of readily available general-purpose RL-RecSys environments for training and bench-marking RL agents. We plan on contributing to this by developing an environment tailored for video recommendations (Netflix, YouTube).

2 Literature Review

In reinforcement learning, the environment serves as the platform where RL agents engage, learn, and improve their decision-making abilities through ongoing interaction and feedback. This feedback comes in the form of rewards or penalties based on the agents' actions. Think of the environment as a virtual playground where RL agents experiment with various methods of recommending items to users, such as products or videos. By receiving feedback on their performance, agents can refine their strategies over time. These environments can simulate real-world scenarios, enabling RL agents to learn and adapt before deploying their strategies in real-world settings.

Building environments demands expertise in simulation and algorithm design. Various general RL environment building resources, such as OpenAI Gym, Microsoft Project Malmo, etc. offer tools, frameworks, and platforms for developing, testing, and benchmarking RL algorithms. These resources contribute significantly to advancing RL research and development by providing standardized toolkits and specialized platforms tailored for environment development and experimentation.

Recommender systems provide personalized recommendations based on the user history, preferences and interests. These systems are implemented in various online platforms influencing the user's decisions and enhancing their experiences while interacting with the system. Some of the RecSys-based environment building frameworks are:

1. **RecoGym**: RecoGym[9] offers tools to create customizable simulations that capture user preferences, item characteristics, and user-item interactions. Various scenarios and pa-

rameters can be explored to analyze the performance of the algorithms under different. RecoGym aims to provide a platform for benchmarking and comparing different recommendation approaches in a controlled setting.

2. **RecSim**: RecSim[2], developed by Google AI, is a configurable simulation platform designed for modeling user behavior, interest evolution, and recommendation strategies within recommender systems. Unlike Reco-Sim, which focuses on creating simulated environments, RecSim provides tools for simulating realistic user interactions and evaluating the effectiveness of recommendation algorithms in dynamic environments. Researchers can use RecSim to gain insights into how different factors influence user engagement and satisfaction, enabling them to improve recommender system performance.

General environment frameworks may lack the specificity required to accurately model the complexities of user-item interactions within recommender systems. RecSim addresses this deficiency by offering a tailored simulation platform designed specifically for modeling user behavior, interest evolution, and recommendation strategies within recommender systems.

In recommender systems research, specialized environments like RecSim play key roles in simulating the recommendation problem. RecSim offers a configurable simulation platform tailored for recommender systems, enabling researchers to create custom environments that mimic diverse user preferences and item characteristics. It facilitates benchmarking of RL algorithms, aiding comparisons and parameter adjustments to enhance algorithm performance.

Building recommender system environments is complex, especially when aiming for accurate modeling of user behavior, item characteristics, and recommendation dynamics. Challenges include capturing diverse user preferences and behaviors, adapting to evolving preferences and item popularity, handling large-scale datasets efficiently, defining suitable evaluation metrics, and addressing data sparsity and cold start issues. Due to the presence of a reliable dataset for video recommendations, we decided to construct a specialized environment within RecSim for this purpose.

The paper[9] survey explores video recommender systems (VRS), which suggest videos to users based on their preferences. The paper examines different VRS approaches, including content-based and collaborative filtering, along with their applications in entertainment and other domains. It also identifies key research challenges in VRS, such as making recommendations when data is limited and ensuring the system can handle large amounts of videos. Overall, this survey provides a comprehensive look at the current state of VRS and highlights promising areas for future research.

In the following section, we go into our environment build for video recommendations, leveraging the above papers for reference.

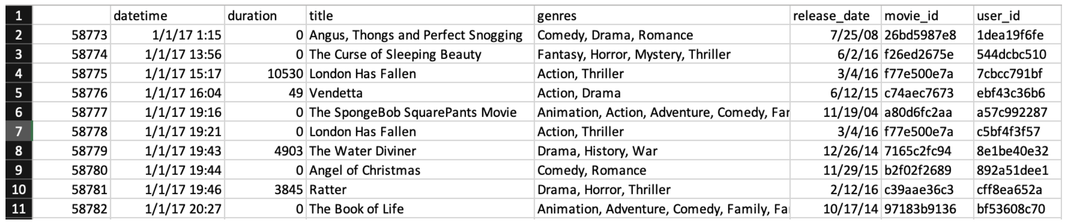
3 Methodology

3.1 Dataset and Feature Engineering

The dataset we use for benchmarking is an audience behavior dataset for Netflix users in the UK, hosted on Kaggle by Vod Clickstream et al. It was collected from users who opted-in to have their anonymized browsing activity tracked. It only includes desktop and laptop activity (which Netflix estimate is around 25(%) of global traffic) and is for a fixed window of time (January 2017 to June 2019, inclusive). It documents each time someone in their tracked panel in the UK clicked on a Netflix.com/watch URL for a movie. Figure below informs about how the dataset looks:

Often referred to as the core of learning-based endeavors, 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 process behind 'engineering' them:



1		datetime	duration	title	genres	release_date	movie_id	user_id
2	58773	1/1/17 1:15	0	Angus, Thongs and Perfect Snogging	Comedy, Drama, Romance	7/25/08	26bd5987e8	1dea19f6fe
3	58774	1/1/17 13:56	0	The Curse of Sleeping Beauty	Fantasy, Horror, Mystery, Thriller	6/2/16	f26ed2675e	544dcbc510
4	58775	1/1/17 15:17	10530	London Has Fallen	Action, Thriller	3/4/16	f77e500e7a	7cbcc791bf
5	58776	1/1/17 16:04	49	Vendetta	Action, Drama	6/12/15	c74aec7673	ebf43c36b6
6	58777	1/1/17 19:16	0	The SpongeBob SquarePants Movie	Animation, Action, Adventure, Comedy, Far	11/19/04	a80d6fc2aa	a57c992287
7	58778	1/1/17 19:21	0	London Has Fallen	Action, Thriller	3/4/16	f77e500e7a	c5bf4f3f57
8	58779	1/1/17 19:43	4903	The Water Diviner	Drama, History, War	12/26/14	7165c2fc94	8e1be40e32
9	58780	1/1/17 19:44	0	Angel of Christmas	Comedy, Romance	11/29/15	b2f02f2689	892a51dee1
10	58781	1/1/17 19:46	3845	Ratter	Drama, Horror, Thriller	2/12/16	c39aae36c3	cff8ea652a
11	58782	1/1/17 20:27	0	The Book of Life	Animation, Adventure, Comedy, Family, Fa	10/17/14	97183b9136	bf53608c70

Figure 2: Netflix Audience Behavior Dataset - UK Movies

- **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.
- **Cluster ID**: Integer values derived from the binary representation of the genre feature vector. These IDs are utilized by RecSim to sample recommendations.
- **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.
- **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.

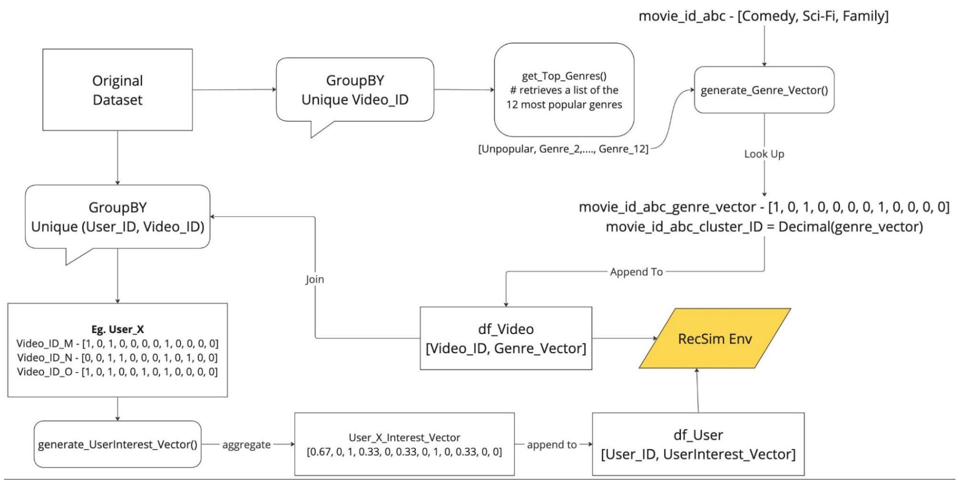

movie_id_abc - [Comedy, Sci-Fi, Family]
Original Dataset
GroupBY Unique Video_ID
get_Top_Genres() # retrieves a list of the 12 most popular genres
generate_Genre_Vector()
[Unpopular, Genre_2,...., Genre_12]
Look Up
GroupBY Unique (User_ID, Video_ID)
movie_id_abc_genre_vector - [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
movie_id_abc_cluster_ID = Decimal(genre_vector)
Append To
Join
Eg. User_X
Video_ID_M - [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
Video_ID_N - [0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0]
Video_ID_O - [1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
df_Video [Video_ID, Genre_Vector]
RecSim Env
generate_UserInterest_Vector()
aggregate
User_X_Interest_Vector [0.67, 0, 1, 0.33, 0, 0.33, 0, 1, 0, 0.33, 0, 0]
append to
df_User [User_ID, UserInterest_Vector]

Figure 3: Steps involved in Feature Engineering

3.2 RecSim

The RecSim platform enables us to simulate recommendation system environments, where the recommendation agent interacts with a set of recommendable documents and users, to help in development of new recommendation algorithms. These environments can be configured to capture specific aspects of the user features like choice models, user preferences and document features, to create natural sequential interactive simulations.

A RecSim simulation can be described by the following event loop which runs for some fixed number of sessions (episodes):

```
for episode in [1,...,number_of_episodes]:
  user = sample_user()
  recommended_slate = null
  while session_not_over:
    user_response = user_responds_to_recommendation(recommended_slate)
    available_documents = sample_documents_from_database()
    recommended_slate = agent_step(available_documents, user_response)
```

Our focus was to create a RecSim environment for Video recommendation systems similar to Netflix. The above loop is realized by implementing the following set of components in RecSim:

- Document Model
- Document Sampler
- User State
- User Sampler
- Response Model
- Choice Model
- User Model
- Reward

3.2.1 Document Model

It inherits from *recsim.document.AbstractDocument* and defines the class that is used to create recommendable document objects in the environment. This class specifies the document features that are observable to agents and are used by agents to evaluate a video object. In our environment we create video objects using the **NetflixVideo** class. It has 2 important properties: features which represent video feature vector and *cluster_id*. The agent receives features as an observable property of a video object.

3.2.2 Document Sampler

A sampler to sample a new set of documents from the document database after each step. The size of this set is defined in the User Model. In our environment, **NetflixVideoSampler** class implements the document sampler. The *sample_document* function of this class returns a random video object from our loaded dataframe.

3.2.3 User State

This class is a container for parameters that define a user's complete state. Additionally, it implements the *score_document* function, which scores the input document w.r.t. The given user. In our environment, **NetflixUserState** class implements the user state. A user has *user_interests* vector and *time_budget* to cap the session length. The *score_document* calculates the relevance score of a document using the dot product of the document's features vector and *user_interests* vector

3.2.4 User Sampler

Sampler to sample a new user in each episode. In our environment, **NetflixUserSampler** class implements the user sampler. The *sample_user* function of this class returns a random user from our loaded dataframe.

3.2.5 Response Mode

One response is generated for each document in the slate. This response is what the agent observes as document-specific user's feedback to the recommendation. In our environment, **NetflixResponseModel** class implements the response model. Each response has the following parameters: *click, watch_time, watch_time_fr, quality, and cluster_id*

3.2.6 Choice Model

The recommended slate presents the user with a set of documents, from which the user can select a document to view or not select anything. This simulation is implemented by the choice model. In our environment, the choice model applies the softmax function to the relevance score given by *score_document* for each document in the slate. The input to softmax also includes a parameter *no_click_mass* that controls the score assigned to not selecting any document. Thus the softmax generates probabilities of each document being clicked by the user and a single probability for not selecting anything. Then the choice model selects the document with the highest probability or nothing.

3.2.7 User Model

The umbrella class that maintains user state, updates the user's state after interaction with the recommendation and generates the user's response to the recommended slate. In our environment, **NetflixUserModel** class implements the user model. The important functions of this class are: *simulate_response* - generates a response for each document in the recommended slate, *update_state* - updates the user's state after the interaction with the slate, and *is_terminal* - checks if the terminating condition of the session is met.

- **simulate_response**: calculates the *watch_time* and *watch_time_fr* as a function of relevance score of video (given by *score_document*), for the clicked video
- **update_state** :
  - decrease user's *time_budget* by the *watch_time* of clicked video
  - updates *user_interest* vector by increasing weights of genres to which the clicked video belongs to. So the user's interests are pushed in the direction of the genres of the selected video.
  - Increase *time_budget* by a factor that's inversely proportional to the relevance score of clicked video w.r.t. user's interest. Emphasizing the fact that the more diverse the recommendation, the more the increase in *time_budget*
- **is_terminal** - check if *time_budget* is greater than 3 min.

3.2.8 Reward

Reward at each time step is the watch time of the clicked video or 0 if nothing was clicked

The watch time is directly proportional to the relevance score. So the reward at each time step is also directly proportional to the relevance of the clicked video, but the time budget is increased by a factor inversely proportional to the relevance score. So, for a greater episodic reward, the agent needs to learn to strike the right balance between how relevant and divergent the recommended videos are w.r.t. evolving the user's interest.

Following figure illustrates how different components work together:

4 Experiment

Within the methodology section, we extensively cover the components of the environment and elucidate their operational functions. Additionally, we delineate how our environment allows longer user engagement if there is a right balance between the relevance and diversity of the recommended videos with respect to user's evolving. To verify the effectiveness of our environment, we benchmarked it by comparing the performance of three agents - random, greedy, and DQN. Below we briefly describe each agent.

4.1 Benchmarking agents

1. **Random Agent**: The *RandomAgent* is a straightforward recommender system agent designed for the RecSim framework. As an implementation of the *AbstractEpisodicRecommenderAgent* class, it recommends a random selection of documents from the available

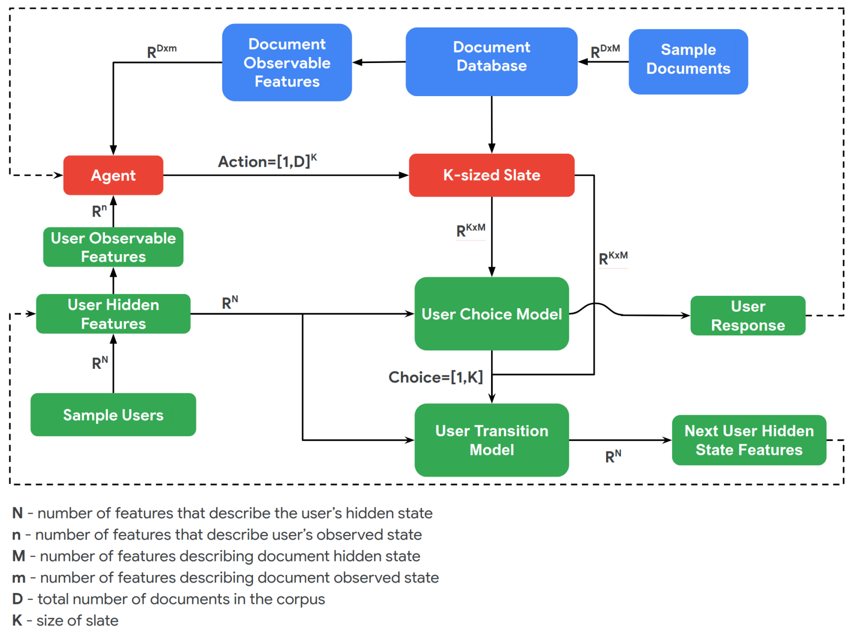

Document Observable Features
Document Database
Sample Documents
R^{Dxm}
R^{DxM}
Agent
Action=$[1,D]^K$
K-sized Slate
R^n
User Observable Features
R^{KxM}
R^{KxM}
User Hidden Features
R^N
User Choice Model
User Response
R^N
Sample Users
Choice=[1,K]
User Transition Model
Next User Hidden State Features
R^N
N - number of features that describe the user's hidden state
n - number of features that describe user's observed state
M - number of features describing document hidden state
m - number of features describing document observed state
D - total number of documents in the corpus
K - size of slate

Figure 4: Interaction of Components

pool. Initialized with a random seed, it utilizes numpy's random number generator to create a randomized slate of documents for each recommendation step. This agent's simplicity makes it a great baseline for comparison with more complex recommendation strategies within the RecSim environment.

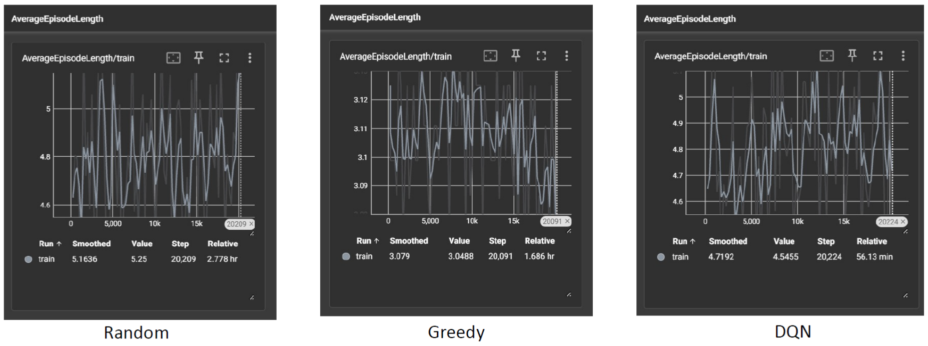
2. **Greedy Agent**: The *GreedyPCTRAgent* is an agent designed for recommendation systems within the RecSim framework. It leverages knowledge of the true underlying choice model to recommend slates with items that have the highest *predicted click-through rate (pCTR)* according to the model. Upon receiving observations of the user and document states, it utilizes a multinomial logit choice model to score documents based on their likelihood of being clicked. Then, it selects the top scoring documents to form the recommendation slate. This agent operates in a myopic fashion, optimizing each recommendation step without considering long-term interactions. Its design enables it to exploit known user preferences to make valid short-term recommendations, making it a valuable component in understanding myopic recommendation strategy within the RecSim environment.
3. **DQN Agent**: The *DQNAgentRecSim* is a specialized Deep Q-Network (DQN) agent tailored for recommendation systems within the RecSim framework. It employs custom adapters to preprocess user and document observations, converting them into a format suitable for neural network input. The agent's neural network architecture, defined in the *recsim_dqn_network* function, concatenates user and document inputs and passes them through multiple dense layers to produce Q-values, informing decision-making. Leveraging the Dopamine library, the agent utilizes a circular replay buffer to store experiences for efficient training. With its ability to adapt to the intricacies of recommendation scenarios, the *DQNAgentRecSim* serves as a powerful tool for enhancing long-term recommendations within the RecSim environment.

4.2 Experiment Description and Expected Interpretations

We trained each agent for 80 episodes i.e. about 20K+ timesteps. To examine the performance of agents we look at the *"Average Episode Length per time-step"*, *"Average Episode Reward per time-step"* and *"Change in User Interest Vector with Time in a single episode"* for all three agents.

For Random and Greedy agents, we expect to observe fluctuating *AverageEpisodeLength* per time-step graphs due to their static policies, which prevent them from learning over time. The random sampling of documents at each time step justifies these fluctuations - since there is no prior learning transferable to freshly sampled documents fluctuations occur. When analyzing the average episode length during training, we expect the Random Agent to a demonstrate fluctuating pattern and relatively constant slope, indicating a lack of learning or improvement over time. For the Greedy agent, we anticipate shorter episode lengths, as it tends to myopically recommend documents aligned with the user's interests, disregarding the exploratory aspect of recommendations to simulate long-term user engagement. Conversely, for the DQN agent, we anticipate higher values of episode lengths and a monotonically increasing graph, reflecting its diversified learning capability. This is attributed to DQN's ability to strike a balance between recommendation alignment and diversity concerning the evolving interests of the user. Since the reward gains are directly proportional to episode lengths, we anticipate the *AverageEpisodeReward* per time-step graphs for three agents will have a similar pattern as corresponding *AverageEpisodeLength* per time-step graphs.

4.3 Experiment Results and Discussions



Random:

Run ↑	Smoothed	Value	Step	Relative
train	5.1636	5.25	20,209	2.778 hr

Greedy:

Run ↑	Smoothed	Value	Step	Relative
train	3.079	3.0488	20,091	1.686 hr

DQN:

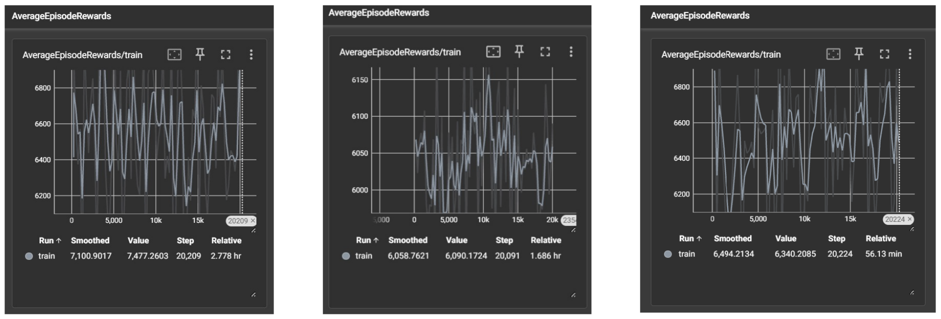
Run ↑	Smoothed	Value	Step	Relative
train	4.7192	4.5455	20,224	56.13 min

Figure 5: Benchmarking Agents (Average Episode Length)

- **Random Agent**: As anticipated, the Random agent demonstrated no discernible learning behavior in both Average Episode Length and Average Episode Rewards graphs. Random Agent's episode length range from 4 to 5 and Average Episode reward fluctuates between 6400 and 6800 with no trend in learning pattern.
- **Greedy Agent**: Aligning with our expectations, myopic recommendations of the Greedy agent lead to smaller increments to user's time budget resulting in shorter episode length on an average - approximately 3 time steps. As the average episode length is smaller, the average episode reward is also smaller - ranging between 6000 and 6100. The greedy agent tried to maximize rewards initially but suffers heavy penalties when similar documents in a recommendation cluster are exhausted.
- **DQN Agent**: The DQN agent displays an increasing learning curve in both the Average Episode Length and Average Episode Reward. DQN agent's average episodic length of 5 is greater than that of Greedy agent's, illustrating the diverse recommendations DQN makes which result in increase in time budget. Also the increasing reward curve demonstrates the exploration vs exploitation approach of DQN which allows the agent to strike a balance between how relevant and how diverse the recommendations are with respect to evolving user's interest. The reward for DQN ranges from 6200 to 6800 which is considerably higher than Greedy agent's range and follows a distinct upward trend in terms of learning than the random agent.

Change in User Interest Vector with Time for 1 episode

The above plots show the change is the user's interest vector as a result of interaction with the recommended slate for a single episode, for all 3 agents. We can observe that Greedy agent concentrates on only 2 user interest genre weights throughout the episode, reflecting its myopic recommendations. This results in a shorter episode length of 3. In contrast to this, the DQN agent's updates to user interest vector are distributed across all genres weights throughout the episode reflecting its diverse recommendations - governed by balanced exploration and exploitation. This results in a longer episodic length of 8. In summary, our comparison underscores how balanced exploration


AverageEpisodeRewards

AverageEpisodeRewards/train

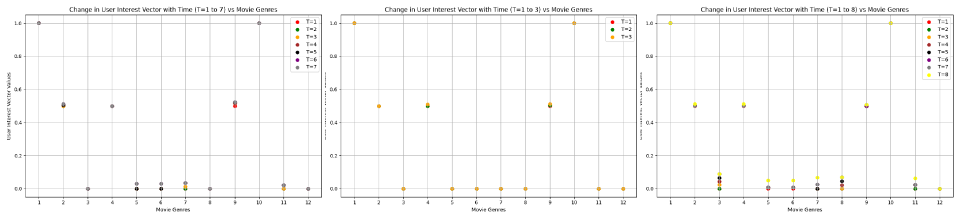
Run ↑	Smoothed	Value	Step	Relative
● train	7,100.9017	7,477.2603	20,209	2.778 hr

Run ↑	Smoothed	Value	Step	Relative
● train	6,058.7621	6,090.1724	20,091	1.686 hr

Run ↑	Smoothed	Value	Step	Relative
● train	6,494.2134	6,340.2085	20,224	56.13 min

Random Greedy DQN

Figure 6: Benchmarking Agents (Average Episode Rewards)



Random Greedy DQN

Figure 7: Benchmarking Agents (Change in User Interest)

and exploitation demonstrated by DQN can achieve longer user session lengths by recommending appropriately relevant documents to the user while maintaining diversity. In conclusion, we observe results that align with our expectations for how the three agents would perform. This inherently indicates our environment's correctness as a video RecSys simulator.

5 Conclusion

This research investigates the potential of reinforcement learning to overcome limitations in deep learning-based recommender systems. Reinforcement learning offers a more interpretable framework to address bias and myopia issues in deep learning techniques, while also providing a mathematical foundation for potentially improving long-term recommendation utility. Our objective was to develop an environment using RecSim, tailored for video recommendations similar to Netflix/YouTube. We began by conducting an extensive literature review to gain insights into how different researchers approached building RL environments in general and for video recommendations.

For benchmarking purposes, we used a publicly accessible audience behavior dataset and performed feature engineering to ensure RL agents receive the most relevant data. Users and documents are sampled to extract hidden features and transform into observable features.

We explored different components of a RecSim environment and focused on targeting long-term user engagement by striking a balance between exploration (trying new videos) and exploitation (recommending user favorites). We verified the effectiveness of our environment by conducting benchmarking, which involved comparing the performance of three agents: random, greedy, and DQN.

To assess the performance of the agents, we examined the "Average Episode Length per time-step," "Average Episode Reward per time-step," and "Change in User Interest Vector with Time in a single episode" for all three agents. The DQN agent, as expected, outperformed both Random and Greedy agents, achieving longer user sessions and exhibiting higher learning effectiveness. Thus, validating the correctness of our environment's build.

In conclusion, this work demonstrates reinforcement learning's potential for making recommender systems more interpretable and robust.

6 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)

[5] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.

[6] Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., ... Hassabis, D. (2016). DeepMind Lab. arXiv preprint arXiv:1612.03801.

[7] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., ... Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.

[8] An overview of video recommender systems: state-of-the-art and research issues

[9] Rohde, D., Bonner, S., Dunlop, T., Vasile, F., and Karatzoglou, A. (2018). RecoGym: A Reinforcement Learning Environment for the Problem of Product Recommendation in Online Advertising. arXiv preprint arXiv:1808.00720.