Topic: RL recommendation algorithms for social media content preferences

Literature review findings

We aim to compare different reinforcement learning (RL) algorithms applied in social media recommendation systems, focusing on their ability to maximize long-term user engagement and adapt to dynamic user preferences. In particular, we are interested in algorithms that can identify user preference shifts, whether due to genuine changes in interest or through exploration-driven discovery of new content. The goal is to better understand the evolution of RL methods used for video recommendation, including their bias and mitigation strategies.

To do this, we use a simulated social media environment with two components: a user model with dynamic preferences and a content model with video features (e.g., category, quality, length). Our goal is to evaluate RL algorithms that can effectively match content to users over time, optimizing long-term rewards such as cumulative watch time and engagement (e.g., likes, shares), while remaining responsive to evolving interests.

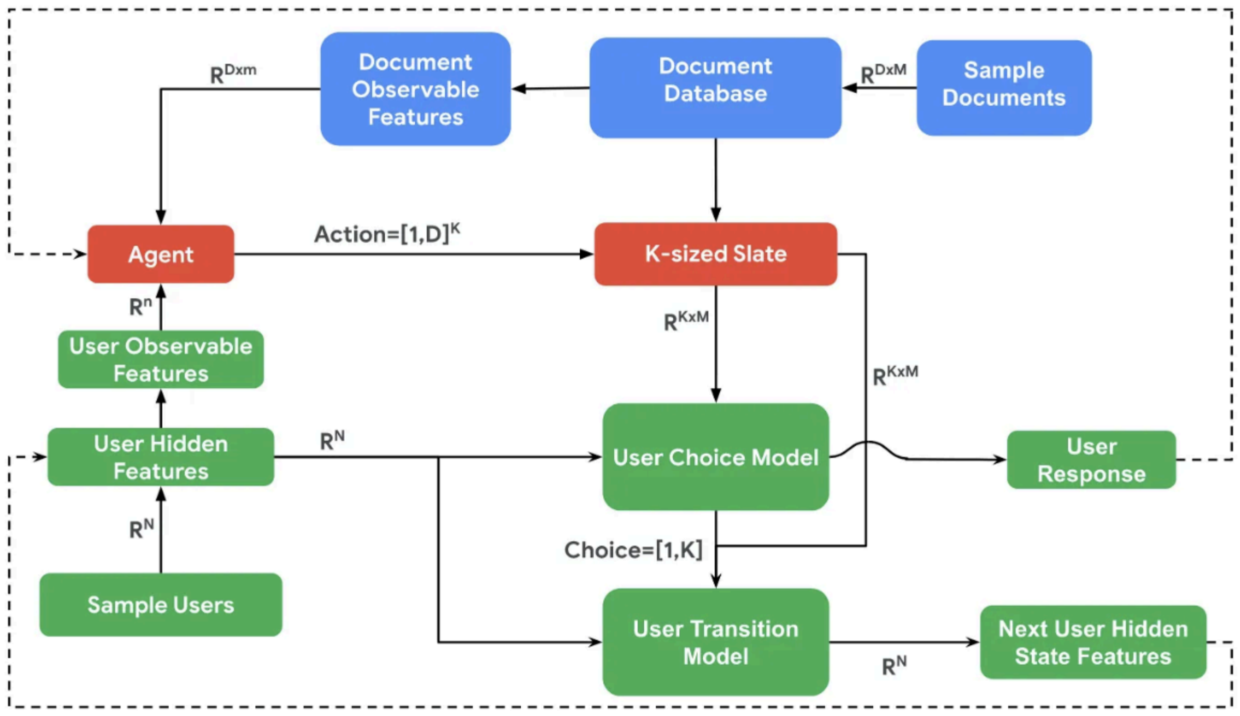
RecSim seems to be a promising tool to simulate, which is a configurable simulation environment that captures preference shifts and user-document interactions. RecSim enables us to experiment with user models that have dynamic interests, as well as documents (e.g., videos) with observable and latent attributes like topic, quality, and length.

To evaluate agent performance, we consider user-centric metrics such as Click-Through Rate (CTR), Watch Time, Completion Rate, Skip Rate, and overall Engagement. These metrics help assess the extent to which different RL algorithms can adapt to preference drift and promote relevant content over time.

Initial implementation results

Our Collaborative Interactive Recommender mostly follows the pipeline of RecSim proposed by Google. However, the original RecSim library is already outdated and only dependent on Tensorflow. Therefore, we adapt the RecSim pipeline to PyTorch and OpenAI Gym, enabling a more lightweight and flexible simulation framework.

[Environment Structure]
We also adapt the case in RecSim, which is a document recommendation, to social media video recommendation.



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

1. Video Sampling and Configuration
   We sampled 10 video candidates per round from a pool of videos. Overall, there are 7 topics available, so within 10 videos, it's likely that some videos are of the same topic and some topics may be not sampled. The recommendation system will recommends a slate of 4 videos at each step

2. User Features
   User features include observable ones, like age and sex; unobservable ones, like personality (2*2 vectors representing the degree of cautiousness and the degree of patience) and their initial interests towards 7 video topics; and some behavioral features, like visit frequency (how many times will the user access the recommender) and time_budget (max clicks per recommendation session).

3. Video features
   Video features include observable ones, like video length, current popularity, topic and history rating; and the latent feature: quality.

4. User choice Model

When the Recommender pushes 4 videos, the users determine whether to click (click probability) based on their own interest, personality and video features. We define that the more cautious the user is, the more he will rely on a video's history rating and popularity to make a click decision. The more patient the user is, the larger length of the video he can tolerate. So, our click probability equals the alignment of user interest and video topic, multiplied by the adaptation term of two personalities.
P(clicked) = dot (user_interest, topic_vector) *(1+cautiousness *(rating + popularity)) * $\exp(\frac{length}{0.1 + patience})$

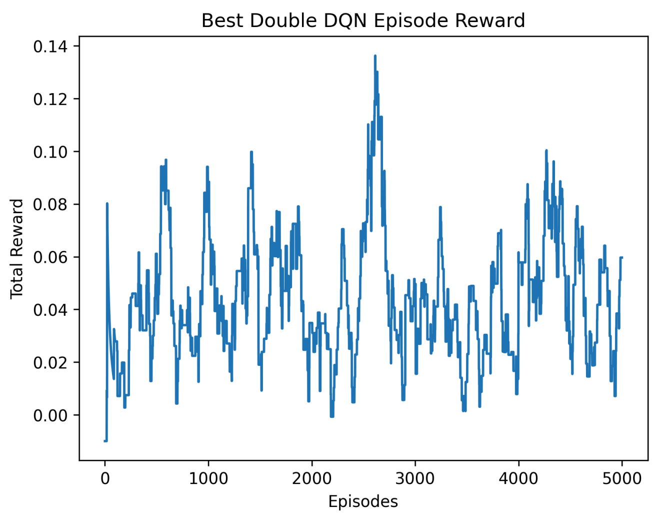
5. User Transition Model
   We define users' latent behavior, and users' interes/preference to specific topics will slowly evolve.
   - Watch time: When a user watches a good video (quality > 0.7), he will watch completely (watch_time =1). Otherwise, the watch time is proportional to the quality of the video (watch_time = quality)
   - Bounce Detection: where a user watches or consumes a video whose quality is below 0.5, the user will bounce, and his interest towards that topic decays by 0.05.
   - Positive Reinforcement: when the user watches 3 videos from the same topic in full, his interest towards that topic will increase by 0.1

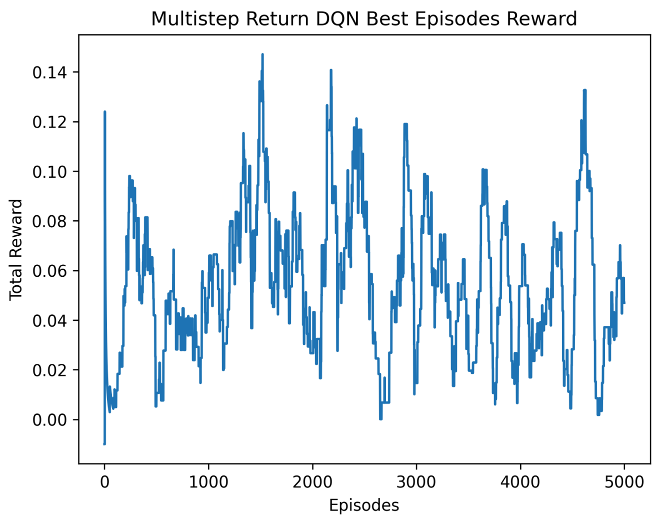
6. Reward Function
   Reward = clicked + α* total_watch_time + γ * bounce_count.
   "clicked" is a bool variable, TRUE =1, FALSE = 0; we set α=1, γ = 0.1

[Agent design & Results]

We selected a series of Deep Reinforcement Learning algorithms as our agent models. Initially, we trained a Double DQN agent and a Multi-step DQN agent. These choices provide more stable Q-value estimation and better propagation of delayed rewards compared to vanilla DQN.

Our DQN implementation was adapted to consider a slate of multiple videos at once, instead of evaluating one video per step. This slate-aware modification allows the agent to learn more complex interactions between user preference and video composition. The result figures are below:





We plan to expand our experiments to include policy-gradient methods such as PPO and A2C, enabling us to compare value-based and policy-based reinforcement learning strategies under the same environment.

Challenges Encounter

[Reward Instability and Low Magnitude]

As shown in the figure above, even if we trained each DQN agent for 5000 epochs, both agents exhibit unstable reward curves with no clear upward trend or convergence. The episode rewards remain low (typically < 0.15) despite the fact that users could potentially click up to 4 videos in a single interaction. Given the reward function, the range of the reward will be around [0,4], even with the adaptation of watch time and bounce rate. We suspected that the initialization of parameters in reward function and the click probability function is not correct, so that the reward might be overly conservative.

So far, Double DQN and Multi-step DQN yield nearly indistinguishable performance in terms of reward progression and variance. This may be attributed to: Insufficient state signal for differentiating learning strategies; Lack of contextual adaptation in reward attribution; and Possibly under-optimized hyperparameters for both agents. We hypothesize that the current environment and reward setup may be over complex or simplified to allow for nuanced agent learning. We plan to address this in the next stage.

Plan for next steps

Our next steps focus on validating and refining the reward pipeline and system dynamics, and testing the environment under different complexity regimes and agent settings.

[Environment Validation and Debugging]

- Re-examine the numerical range and scaling of click probabilities, state transitions, and reward terms to ensure they align with expected user behavior and empirical plausibility.
- Debug whether transition dynamics (e.g., preference decay or reinforcement) are applied correctly and consistently over time.
- Monitor whether interest updates and bounce penalties have meaningful impact on future recommendations.

[Ablation Studies]

- Remove or isolate components like preference shift and user-video feature interaction to observe their contribution to agent learning.
- Increase the number of topics or document diversity to explore whether the current agents can handle higher environment complexity.
- Vary document presentation features (e.g., include noise in popularity) to test the robustness of choice modeling.

[Expanding RL Algorithm Benchmarks]

- Add baseline models such as Multi-Armed Bandits and Contextual Bandits to understand performance in low-complexity regimes.
- Experiment with more advanced policy optimization methods such as PPO, A2C, and Actor-Critic frameworks.
- Use both cumulative reward and reward stability as metrics to evaluate algorithm robustness across different environmental assumptions.

These steps will allow us to better understand how algorithmic strategies adapt to changes in environment design and user modeling assumptions.