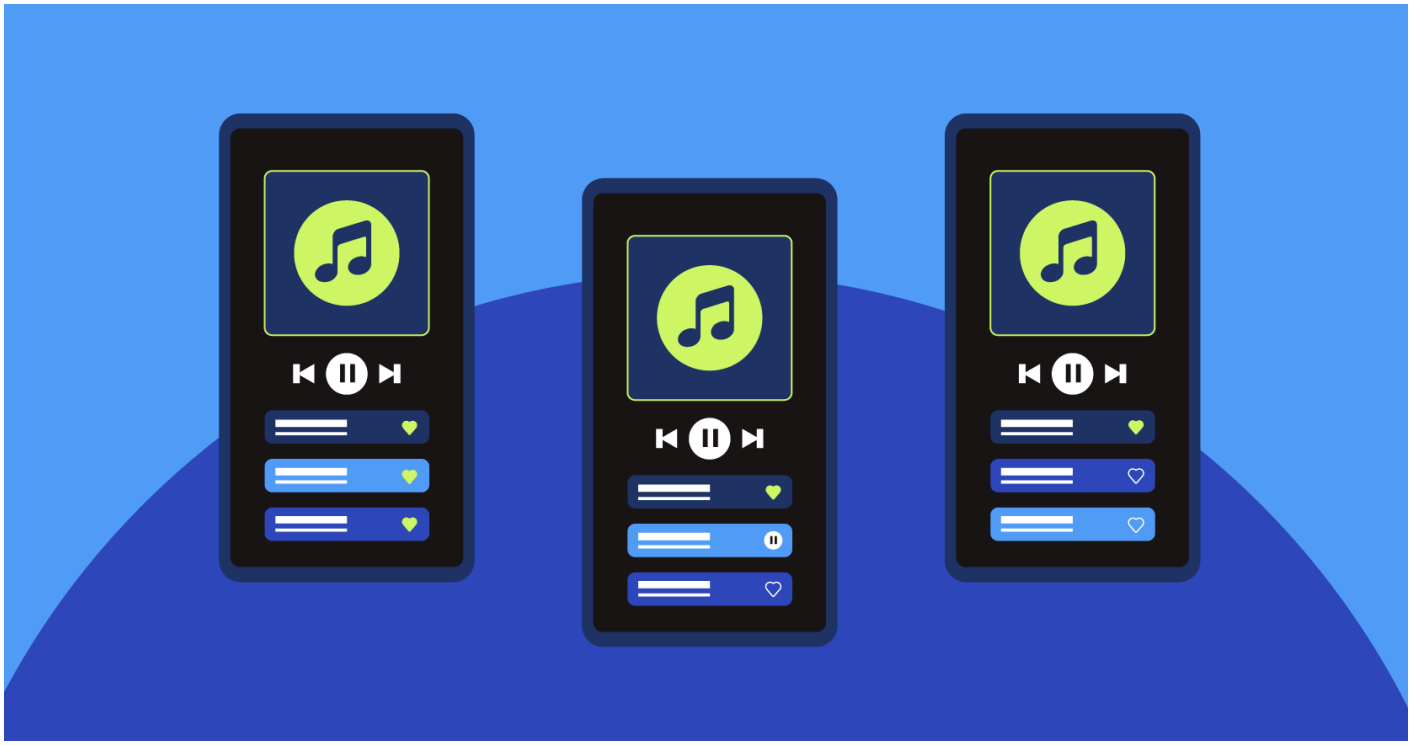
Automatic Music Playlist Generation via Simulation-based Reinforcement Learning

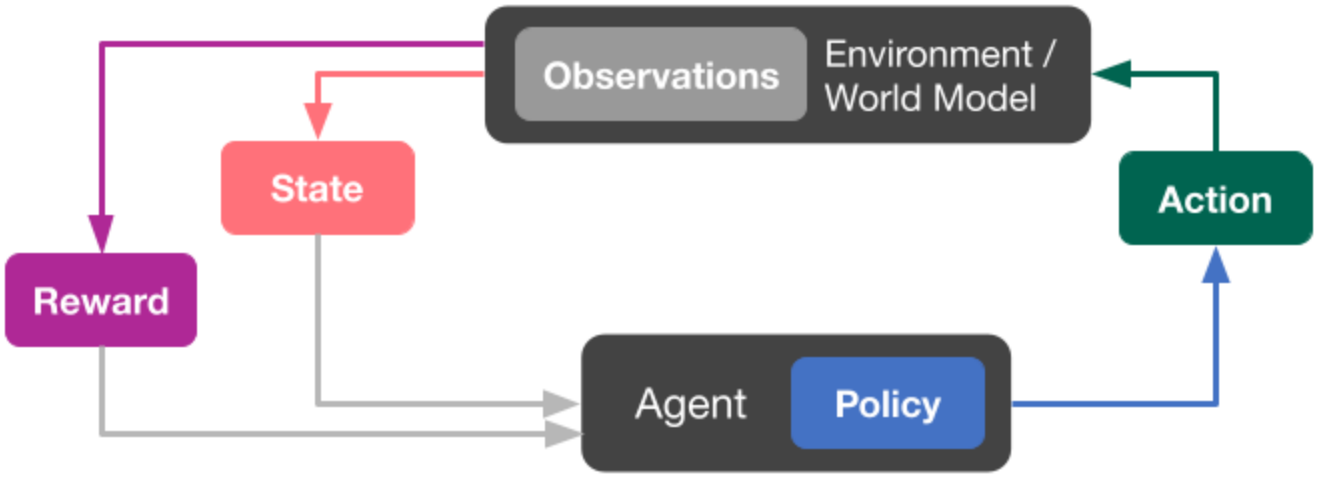
July 19, 2023
Published by Federico Tomasi, Joseph Cauteruccio, Surya Kanoria, Kamil Ciosek, Matteo Rinaldi, and Zhenwen Dai



Reinforcement learning (RL) is an established tool for sequential decision making. In this work, we apply RL to solve an automatic music playlist generation problem. In particular, we developed a RL framework for set sequencing that optimizes for user satisfaction metrics via the use of a simulated playlist-generation environment. Using this simulator we develop and train a modified deep Q-Network, which we call the Action-Head DQN (AH-DQN), in a manner that addresses the challenges imposed by the large state and action space of our RL formulation. We analyze and evaluate agents offline via simulations that use environment models trained on both public and proprietary streaming datasets. We show how these agents lead to better user-satisfaction metrics compared to baseline methods during online A/B tests. Finally, we demonstrate that performance assessments produced from our simulator are strongly correlated with observed online metric results.

We frame the problem as an automatic music playlist generation: given a (large) set of tracks, we want to learn how to create one optimal playlist to recommend to the user in order to maximize satisfaction metrics. Crucially, our use case is different from standard slate recommendation tasks, where usually the target is to select at maximum one item in the sequence. Here, instead, we assume we have a user-generated response for multiple items in the slate, making slate recommendation systems not directly applicable.

As an example, consider the case when users decide on a type of content they are interested in (eg, “indie pop”). Having a catalog of millions of tracks, it is not straightforward which tracks are best suited for the user that requested the playlist, as each user experiences music differently. For this reason, automatic playlist generation is a relevant and practical problem for music streaming platforms to create the best personalized experience for each user.

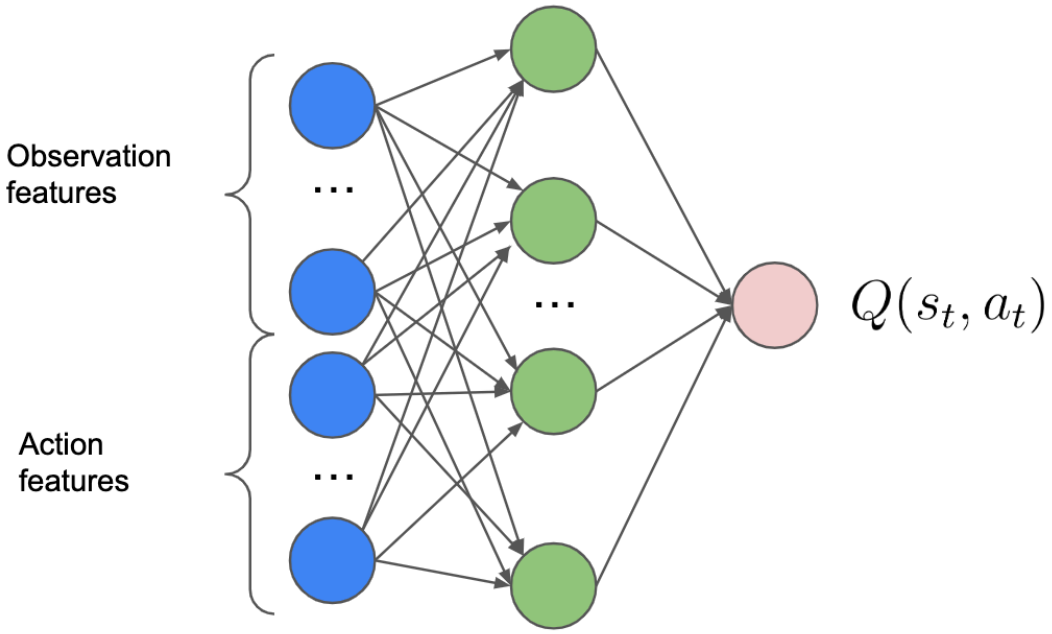


To take into account the constraints and the sequential nature of music listening, we use a reinforcement learning approach. To avoid letting an untrained agent interact with real users (with the potential of hurting user satisfaction in the exploration process), we make use of a model-based RL approach.

In model-based RL, the agent is not trained online against real users. Instead, it makes use of a user simulator, a model that estimates how a user would respond to a list of tracks picked by the agent. Using this model we can optimize the selection of tracks in such a way as to maximize a (simulated) user satisfaction metric. During the training phase the environment makes use of this user model to return a predicted user response for the action recommended by the agent. The user model

The figure above shows the standard reinforcement learning loop, where the action, in playlist generation tasks, is the track to be recommended. The environment consumes the action proposed by the agent and uses the world model (a user simulator) to transition to the new state and returns a specific reward conditioned on the action; for example, whether the user plays the track. The agent is able to see the new state and reward, using them to adapt the policy and predict the next action to pass to the environment for the next iteration. This procedure continues until the world model signals the termination of a specific sequence of states (called episode).

We developed an action head (AH) DQN agent that is able to deal with dynamic candidate pools. In particular, this is a variant of the popular DQN agent. DQN uses a deep neural network to predict the recommendation quality (Q) of each item (action) in a listening session. The main idea behind the Q network is that each available track is assigned to a specific value (a Q value), and the track with the highest value is then selected by the agent. The reward as returned by the environment after each action is used to update the Q network.



Our AH-DQN network takes as input the current state and the list of feasible actions. The network will produce a single Q value for each action in input, and the one with the highest Q value is selected as the next action to apply.

We tested our approach both offline and online at scale to assess the ability of the agent to power our real-world recommender systems. Our recommender system aims at maximizing user satisfaction and, as a proxy, we consider the completion count (and rate) at the session level, i.e., the amount of tracks recommended by the policy that are completed by the user. For this task we first train an agent offline using a non-sequential world model (CWM) as user simulator, and then deploy the agent online to serve recommendations to the users.

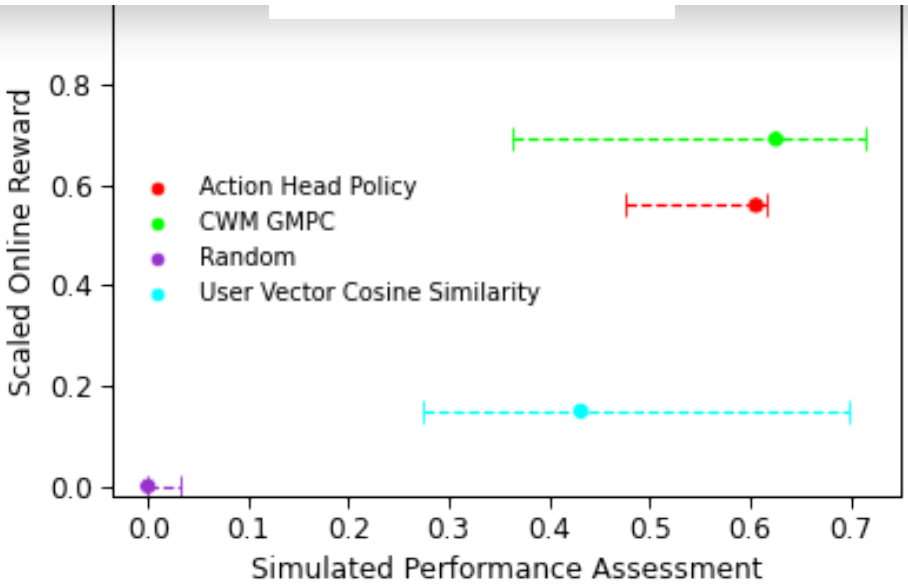
The outcomes of our user model are primarily consumption-focused and summarize the probability of user-item interactions. Specifically, CWM has been optimized for three targets: completion, skip and listening duration. The reward for the agent is computed as the sum of the probability of completion for each track in an episode.

We compare our proposed agent against three alternative policies:

- Random: a model that randomly sorts the tracks with uniform distribution (ie, all tracks have equal probability of appearing in a specific position);
- Cosine Similarity: a model that sorts the tracks based on the cosine similarity between predefined user and track embeddings;
- CWM-GMPC: the user model ranking, which sorts the tracks based on the predicted probability of completion from the user.

All policies take as inputs features of the user who made the request and the pool-provided set of tracks with their associated features. The goal of each policy is to select and order a list of tracks from the pool that maximizes expected user satisfaction, which we measure by counting the number of tracks completed by the user. We conducted an A/B test comparing the previously described 4 policies to the production playlist generation model.

Offline-online correlation



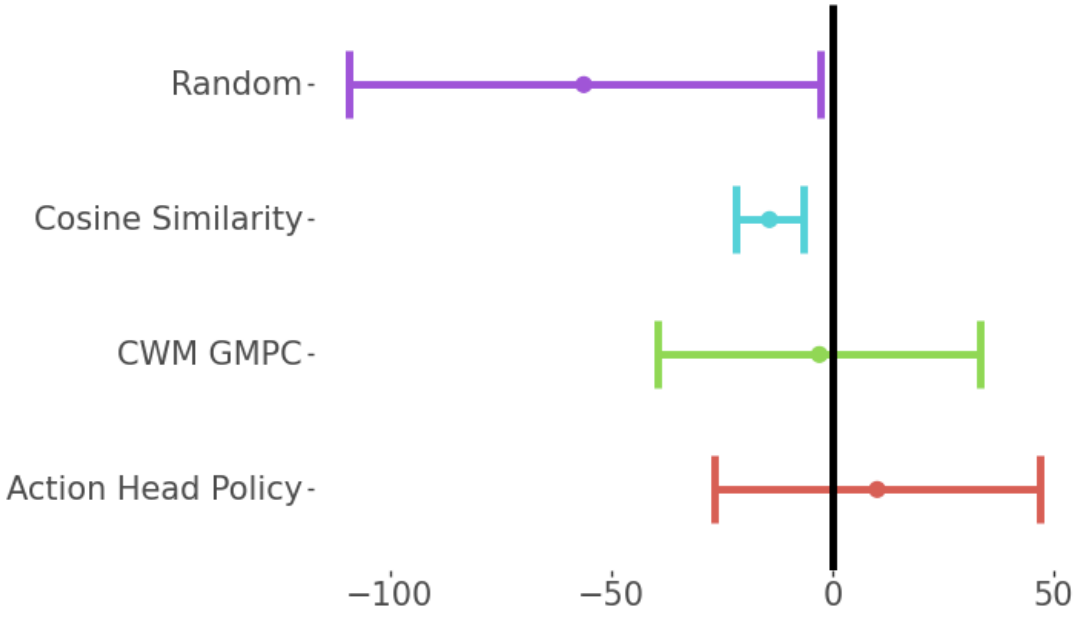
We use an independent metric model in offline simulation to estimate policy performance, the sequential world model (SWM). This is used to estimate offline performance for each model, which we plot alongside online results for the policies previously listed. The offline performance expectations of the evaluated policies align with their online performance.

We see that the offline performance between the Action Head Policy (the agent) and CWM-GMPC is essentially the same. This is expected since the optimal policy for an agent trained against the pointwise world model is, in fact, the greedy policy CWM-GMPC. Online results show a slight gap between these policies, but the difference is statistically indistinguishable.

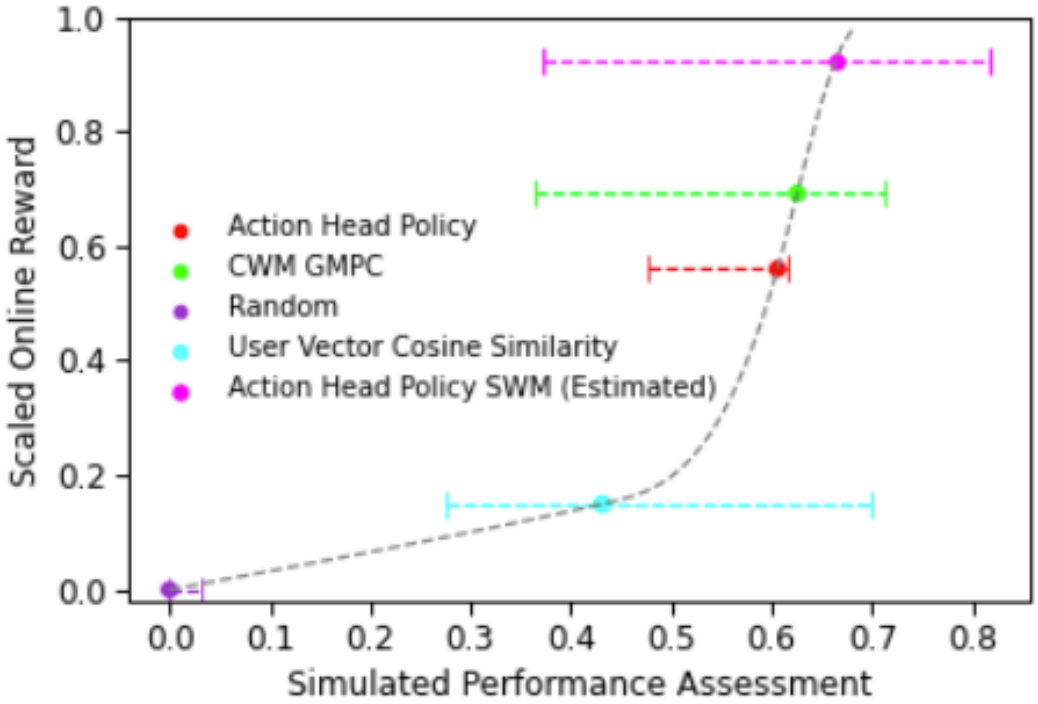
Table 3: Relative percent difference between agent policy and control on online evaluation.

Metric (Per-Session Average):	Relative % Difference:
Completion-Count	10.17 (p: 0.59, CI: -26.79 | 47.13)
Total MSP	6.43 (p: 0.73, CI: -30.67 | 43.53)
MSP-Per-Item	-5.62 (p: 0.46, CI: -20.69 | 9.44)
Skip-Rate	-7.8 (p: 0.33, CI: -23.52 | 7.92)
Completion-Rate	5.39 (p: 0.6, CI: -14.8 | 25.58)
Session Length (interactions)	8.87 (p: 0.53, CI: -19.02 | 36.75)

between the user model and control. This, along with the offline-online correlation analysis, shows how our approach to model user behavior is accurate in practice.



During further evaluation results, we see how both our user model ranking and the agent trained in simulation show results statistically indistinguishable from control.



Finally, we also used an improved user simulator (SWM) to estimate offline results. Based on our previous offline-online correlation analysis, we can estimate how the

improvement that could be translated online. Based on these predictions we hypothesize an online improvement over both CWM-GMPC and AH-CWM policy performance. We argue this type of analysis of being relevant in practical applications of recommender systems where online deployment requires sufficient expected improvement over existing baselines.

Some final words

In this work we presented a reinforcement learning framework based on a simulated environment that we deployed in practice to efficiently solve a music playlist generation problem. We presented our use case which is different from standard slate recommendation tasks where usually the target is to select at maximum one item in the sequence. Here, instead, we assume we have a user-generated response for multiple items in the slate, making slate recommendation systems not directly applicable. The use of RL also enables the easy implementation of additional satisfaction metrics and user signals, in such a way that the work can be easily ported to a wide range of music recommendations problems.

For more information please check out our paper below:
Automatic Music Playlist Generation via Simulation-based Reinforcement Learning
Federico Tomasi, Joseph Cauteruccio, Surya Kanoria, Kamil Ciosek, Matteo Rinaldi, and Zhenwen Dai
KDD 2023

SHARE

CATEGORIES

Machine Learning Search & Recommendations User Modeling

Related articles

PODTILE: Facilitating Podcast Episode Browsing with Auto-generated Chapters

Socially-Motivated Music Recommendation

Sign up for research updates

By clicking sign up you'll receive occasional emails from Spotify. You always have the choice to adjust your interest settings or unsubscribe.

Your Email

Sign Up





Newsroom Spotify Jobs Spotify.com

Spotify R&D Engineering Spotify R&D Design

Legal Privacy Cookies About Ads