Minimizing Live Experiments in Recommender Systems: User Simulation to Evaluate Preference Elicitation Policies

Chih-Wei Hsu
Google Research
Mountain View, USA

Martin Mladenov
Google Research
Mountain View, USA

Ofer Meshi
Google Research
Mountain View, USA

James Pine
Google Research
Mountain View, USA

Hubert Pham
Google Research
Mountain View, USA

Shane Li
Google
Mountain View, USA

Xujian Liang
Google
Mountain View, USA

Anton Polishko
Google
Mountain View, USA

Li Yang
Google Research
Mountain View, USA

Ben Scheetz
YouTube
New York, USA

Craig Boutilier
Google Research
Mountain View, USA

ABSTRACT

Evaluation of policies in recommender systems typically involves A/B live experiments on real users to assess a new policy's impact on relevant metrics. This "gold standard" comes at a high cost, however, in terms of cycle time, user cost, and potential user retention. In developing policies for *onboarding* users, these costs can be especially problematic, since on-boarding occurs only once. In this work, we describe a simulation methodology used to augment (and reduce) the use of live experiments. We illustrate its deployment for the evaluation of *preference elicitation* algorithms used to onboard new users of the YouTube Music platform. By developing counterfactually robust user behavior models, and a simulation service that couples such models with production infrastructure, we can test new algorithms in a way that reliably predicts their performance on key metrics when deployed live.

CCS CONCEPTS

• **Computing methodologies** → **Simulation evaluation**.

KEYWORDS

User modeling, preference elicitation, recommenders, simulation

ACM Reference Format:
Chih-Wei Hsu, Martin Mladenov, Ofer Meshi, James Pine, Hubert Pham, Shane Li, Xujian Liang, Anton Polishko, Li Yang, Ben Scheetz, and Craig Boutilier. 2024. Minimizing Live Experiments in Recommender Systems: User Simulation to Evaluate Preference Elicitation Policies. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24), July 14–18, 2024, Washington, DC, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3626772.3661358

SIGIR '24, July 14–18, 2024, Washington, DC, USA

ACM ISBN 979-8-4007-0431-4/24/07.
https://doi.org/10.1145/3626772.3661358

1 INTRODUCTION

The scale and complexity of modern *recommender systems (RSs)* means they are typically developed with a continuous delivery methodology, where new features are produced, evaluated, and deployed constantly. The gold standard for evaluation of any new feature is controlled A/B tests, using *live experiments (LEs)* with real users, to assess its impact on key long-term RS metrics. The use of LEs for A/B testing, however, has well-known costs [20]. Online evaluation can impose reputational and retention costs by degrading the experience of users exposed to poorly performing (experimental) policies. Statistically significant results often require large sample sizes and long evaluation periods to emerge, which can slow the algorithm development cycle. Finally, opportunity cost is especially acute when testing new *onboarding methods*, a focus in this work: since users onboard only once, a poor user experience may leave no opportunity for recovery.

To alleviate these concerns, *simulation* has been used increasingly for RS research and algorithm development [1, 11, 17, 22, 23]. Simulated *user models* allow developers to "observe" how (stochastic) user responses unfold in reaction to sequences of RS actions, circumventing the out-of-distribution issues of static data sets, allowing effective assessment of new algorithms, and supporting quicker iteration without imposing the costs of LEs on users. That said, simulation is largely used for algorithm development rather than to replace LEs since simulation models: (a) are not completely realistic, (b) are often designed to predict short-term effects rather than the long-term metrics often assessed by LEs, and (c) are rarely connected directly to production infrastructure (which links *other vital RS components* that influence user behavior).

In this work, we propose a practical methodology for the use of simulation not only to help develop new RS algorithms, but to move a step closer to reducing (even partially replacing) LEs for evaluation. We illustrate this methodology using the development, evaluation and deployment of novel onboarding algorithms for new users of YouTube Music. Along the way we outline: (a) the development of user models—in our case recurrent neural networks

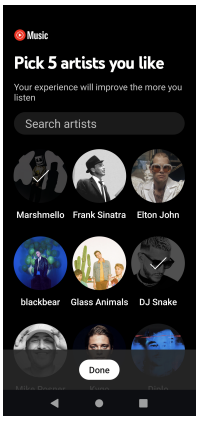

Figure 1: The artist selection interface. Users select artists they like, and skip those they don't, with "Done" confirming the end of the onboarding session. The artists displayed as the user scrolls down are chosen dynamically given earlier selections and non-selections (or *skips*).

[8]—that capture the distribution of (relevant properties of) real users; (b) a process that ensures these models are reasonably counterfactually robust so performance predictions for *new* policies can guide deployment decisions [11]; and (c) the framework used to run our simulated users on production infrastructure to ensure simulated metrics reflect *all* product intricacies.

2 PROBLEM STATEMENT

We first outline the *onboarding process (OP)* for new users of YouTube Music, and the *preference elicitation (PE)* algorithms we have developed to drive user interaction. We refer to Meshi et al. [15] for a detailed description of the PE methods themselves.

The interface used to elicit artist preferences is shown in Figure 1. The user can scroll to select as many artists as desired and choose "Done" to terminate the onboarding process. The PE strategy plays a critical role in the user experience: it impacts *how much* and *what type* of preference information the RS obtains, hence, downstream recommendation quality and usage. If the PE algorithm shows many similar artists to the user, or artists they are not familiar with, the user might decide that onboarding is not worth their time, and abandon before providing much (or any) preference information.

These considerations mean that developing a good PE algorithm for onboarding is highly non-trivial, involving tradeoffs between familiarity and novelty, coverage and "deep dives," and much more. We do not provide detailed algorithm descriptions here other than to point out that our PE methods are both personalized to user context (see below) and dynamic (i.e., artists are chosen adaptively based on a user's previous selections/skips during the onboarding session). For our purposes, it is sufficient to treat various algorithms and their parameterizations as black boxes to be evaluated or compared.[1]

3 METHODOLOGY

To evaluate PE policies for onboarding, we employ simulation as an alternative to costly A/B testing with LEs. Simulation can mitigate the potential cost imposed on users—which is magnified due to the one-off nature of onboarding—and accelerate the development cycle for novel PE policies. This approach aligns with recent advances in simulation usage in RSs, encompassing algorithm development [11, 19, 22, 23], user dynamics analysis [1], and the modeling of sequential [5, 10, 25, 26] and multiagent interactions [16].

[1]For further discussion of onboarding techniques for RSs, see [14], and PE, see [2, 3, 21].

Effective simulation for algorithm development requires synthetic *user models* that accurately represent the distribution of user characteristics and their responses to RS actions to determine algorithm performance when tested with a simulated user population. In our case, user models reflect (conditional) user responses to questions about artist preferences, their chance of exiting the onboarding process, and the distribution of such behaviors in the new-user population. Developing such user models is challenging. Typically, data from an existing RS policy is used to train predictive user models. However, this offline approach is often unable to predict responses to *novel policies*, since new policies induce out-of-distribution behaviors/data w.r.t. the production policy. Without methods for *off-policy evaluation* [6], these models are unlikely to be *counterfactually robust*. We approach this by building models using data from *several* policies that capture a range of user behaviors.

Our user models, which engage in the PE OP, have two requirements: generating a *synthetic user population* whose distribution of relevant characteristics reflects that of the user base; and providing a *user response model* that accurately captures user behavior with our PE methods conditioned on these characteristics. We make two assumptions that support our modeling approach. First, some version of the OP has been deployed prior to the development of new methods targeted by simulation. Second, most users who engaged with the original OP moved on to use the RS for an extended period. This gives a set of users $\mathcal{U}$: for each $u \in \mathcal{U}$ we have both *OP session data* S^u and *post-onboarding usage data*, which we morph into static user state (or *context*) C^u (see details below).

The new OP being tested is driven by one of several PE algorithms being considered for evaluation. These algorithms adapt their artist selection to the previous selections/skips of the user.[2] When presented with an artist, a user makes two distinct binary choices: (i) select or skip that artist; and (ii) terminate or continue with the OP. A simulated user must generate these two choices at each step. Action sampling is conditioned on (a) the *static (or prior) user state*; and (b) the user's OP *session history* so far. The interaction of a synthetic user u with the OP proceeds through a sequence of $t > 0$ turns (until u terminates) as follows: (1) RS (PE module) queries u with an artist. (2) u responds (selects/skips, terminates/continues). (3) u's context (latent state) is updated. (4) RS updates its beliefs about u given response (a black box).

User State Generator. The first component of our generative user model comprises a distribution $P(C^u)$ of *static user states*, those characteristics that drive a user's behavior when engaging with the OP. Our *user state generator* divides user state into *observable context* C^u_o and *latent state* C^u_ℓ. Observable context consists of factors that are directly observable and (potentially) influence a user's musical tastes and OP interactions—these might include geography, device type (e.g., mobile vs. desktop, OS type), and other such features.

Perhaps more critical is the user's latent state, which includes the user's *true* but unobservable underlying musical preferences. There is no data for new users, but by assumption, we have a large number of users who have used the original OP and subsequently engaged with the RS. This allows us to estimate the true preferences of such users, *retrospectively* correlate them with behavior during the OP,

[2]Our PE methods are also personalized given prior music engagement by the user, but we set this aside in our treatment here.

and generate realistic preference distributions. Together with the observable state, this comprises our *user state generator*. Specifically, we augment OP session data with an ordered list $\{q_1^u, q_2^u, \ldots, q_K^u\}$ comprising the top-K artists w.r.t. u's post-onboarding music listening time during the first W weeks of YouTube Music usage.[3] We use this data to train latent user-preference models.

To train the user state generator $P(C^u)$, we can easily learn a categorical (joint) distribution over relevant observable state variables C_o^u from our data. However, directly modeling the joint distribution $P(C_\ell^u)$ of latent preferences (i.e., all combinations of inferred user interests) is intractable. Hence, we formulate our latent state model as a sequence generation or *next-item prediction* problem [24] conditioned on the observable state c_u. Specifically, given a user's (post-OP) music consumption, we predict the next item q_k^u given a user u's past interaction history: $P(q_k^u|q_1^u, \ldots, q_{k-1}^u)$. We factorize the joint distribution $P(C^u)$ as:

$$P(C^u) = P(c^u, q_1^u, \ldots, q_K^u) = P(c^u) \prod_{k=1}^{K} P(q_k^u | c^u, q_1^u, \ldots, q_{k-1}^u). \quad (1)$$

Recurrent neural networks (RNNs) [9] have proven effective for next-item prediction [7, 12, 18]. Thus, we use an RNN to model the temporal dependencies needed to generate inferred user interests $q_1^u, \ldots, q_K^u$. We apply a trainable multilayer perceptron (MLP) adapter to the one-hot encoding of c^u and seed the initial state of the RNN h_0^u with the output. The RNN updates its hidden state h_k^u and generates output o_k^u: $(h_k^u, o_k^u) = RNN(h_{k-1}^u, \phi(q_k^u), \delta_k)$ where $\phi(q_k^u)$ is the embedding of artist q_k^u—we reuse embeddings from the PE module—and δ_k is a one-hot encoding of the number of artists generated so far. The next artist q_{k+1}^u is then sampled (without replacement) using a *multinomial logit* model [4, 13] given o_k^u:

$$P(q_{k+1}^u | o_k^u) = \frac{\exp(\phi(q_{k+1}^u)^\top o_k^u)}{\sum_{q \in Q \setminus \{q_i^u\}_{i \le k}} \exp(\phi(q)^\top o_k^u) + \exp(\phi_\varnothing^\top o_k^u)}. \quad (2)$$

To generate variable-length sequences, we introduce a fixed *null artist* embedding $\phi_\varnothing$. We do not increase k or change the input vector when the null artist is sampled. Including δ_k as an RNN input helps match the distribution of the number of artists in users' inferred interests to the observed data.

Our latent state generator is implemented with a *long short-term memory (LSTM)* architecture [8]. By modeling generation of user interests $P(q_{k+1}^u | C^u, q_1^u, \ldots, q_k^u) = P(q_{k+1}^u | o_k^u)$ with an RNN and $P(c^u)$ as a categorical distribution, we can optimize parameters of the entire generator by maximum likelihood estimation (MLE).

User Session Generator. The second component of our model is a *user session generator* that samples a user's action choice when presented with an artist, conditioned on their (static) state and their action choices thus far. Users' attitudes evolve during the OP (e.g., degree of engagement/frustration with the OP, or satisfaction with the information provided). These are part of the user's *latent, dynamic state* implicitly captured in our model.

To train the session generator, we use a set of artists $Q = \{q_1, \ldots, q_m\}$ eligible to be shown during PE. A session S^u for $u \in \mathcal{U}$ comprises a sequence of artists/queries q_t^u and associated responses r_t^u: $S^u = \{(q_1^u, r_1^u), \ldots, (q_{|S^u|}^u, r_{|S^u|}^u)\}$, where $r_t^u \in \{0, 1\}$ is the user response (skip/select) to q_t^u at turn t. User termination occurs only at turn $t = |S^u|$. Let $S_{<t}^u = \{(q_1^u, r_1^u), \ldots, (q_{t-1}^u, r_{t-1}^u)\}$ be u's subsession prior to t. This format assumes users inspect artists linearly, which may be only roughly valid in UIs where multiple artists are visible at once. We truncate sessions at 300 turns.

We formulate the user response model $P(r_t^u | C^u, S_{<t}^u, q_t^u)$ as contextual sequence generation, using an RNN to encode *(in-session) dynamic* user state and its dynamics. We first embed inferred user interests in C^u via function ϕ, to which we apply a trainable MLP to obtain encoded context $E(C^u)$ to personalize session state, dynamics and output (user response) for a synthetic user u. The input vector x_t^u to the RNN at turn t includes $E(C^u)$, the embedding $\phi(q_t^u)$ of the artist/query, the embedding of selection response $\phi^r(r_t^u)$, the number y_t of selections so far, and t. The latter two are critical for reproducing realistic distributions of the number of artist selections and session length. The response embedding ϕ^r is trainable but, as above, we reuse artist embeddings $\{\phi(q)\}_{q \in Q}$ from PE. The RNN hidden state h_t^u at turn t is updated as: $(h_t^u, o_t^u) = RNN(h_{t-1}^u, x_t^u)$. where $h_0^u = E(C^u)$ and o_t^u is the output of the RNN at turn t.

OP sessions S^u vary in length since u determines when to terminate given their latent state and the current artist. Since we must also predict session continuation s_t^u, we use an MLP with two heads and sigmoid activation σ for r_t^u and s_t^u:

$$P(r_t^u = 1 | C^u, S_{<t}^u, q_t^u) = \sigma(MLP(o_{t-1}^u, \bar{x}_{t-1}^u)) \cdot s_{t-1}^u, \quad (3)$$

$$P(s_t^u = 1 | C^u, S_{<t}^u, q_t^u) = \sigma(MLP(o_{t-1}^u, \bar{x}_{t-1}^u)) \cdot s_{t-1}^u. \quad (4)$$

We also pass $\bar{x}_{t-1}^u = (E(C^u), y_{t-1}, t-1, \phi(q_t^u))$ to the MLP. The response model does not predict a selection if the model predicts termination. As above, the session generator is implemented using an LSTM, and we optimize its parameters by MLE.

The Simulator. Evaluating *actual deployment* of an algorithm requires integration into a production stack. Likewise, truly effective use of simulation requires running the *production RS* with simulated users. We integrate synthetic user models into the production RS (vs. a mock RS) using a service that injects simulated user interactions into the full content-serving stack. Simulated-user data is segregated from real-user data, but is otherwise indistinguishable to the production RS.

Our simulator interacts with a front-end service that mimics the YouTube Music app, and with our user models. For each simulated user, a test account is created and user data is initialized. The simulator presents artists used for PE from the front-end in a linear fashion. User-selected artists are submitted to the RS and stored so as to affect downstream recommendations. Use of a single *user data serving service (UDSS)* ensures consistency and proper access control of (real and synthetic) user data for dependent systems. A *Data Overlay Service (DOS)* allows reads and writes of synthetic data for the RS. When UDSS receives requests from the simulator, it retrieves data from the DOS, replacing or merging it with production data.

4 EXPERIMENTAL RESULTS

We train RNN user models with logged data from YouTube Music OP sessions, using the user-simulation platform RecSim NG [17].

[3] Initial RS recommendations will of course be correlated with onboarding selections and affect post-onboarding consumption. However, user searches and RS exploration will make this effect smaller as the horizon W increases.

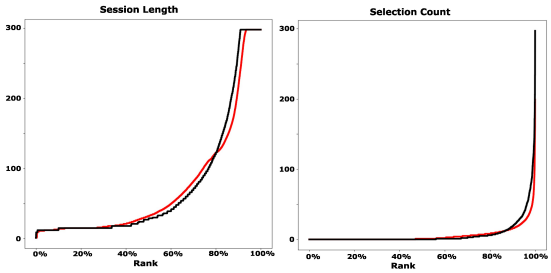

Figure 2: Cumulative distribution functions (CDFs) for both session length (left) and selection counts per session (right). Generated CDFs are red, observed CDFs black.

	Users	Selections	Impressions
Live Exp.	~134 000	-0.76% [-5.05%, 3.04%]	-1.25% [-3.57%, 1.07%]
Simulation	200 000	+1.36% [-0.11%, 2.83%]	-1.02% [-1.74%, -0.31%]
Post-launch	500 000	+1.17% [-0.03%, 2.36%]	-0.78% [-1.15%, -0.41%]

Table 1: Comparison of Live Experiment, Simulated, and Post-launch Metrics of New pCTR Policy on a Traffic Slice.

We deploy our simulation service for the OP so simulated user models interact with the production RS. We use a well-performing OP as the data-generating policy for user-model training and evaluate our models based on their ability to reproduce key statistics from logged (real) user (test) data. As our static user-state generator is policy-independent, we compare its generated context distribution with the ground truth. For space reasons, we omit results with *existing* PE policies, and instead emphasize our model results when interacting with *previously unseen* PE policies, focusing on counterfactual robustness and policy optimization via offline simulation.

Simulation of New Policies: To evaluate the counterfactual robustness of our user models, we simulate their interactions with unseen OP PE policies. Specifically, we deployed a new policy that prioritizes artists with higher predicted click-through rates (pCTR). Using 200,000 synthetic users (from one geo, one device type), we forecast the policy's impact on impressions and selections. For validation, we also conducted an LE with 134,000 real users (same geo, device type). Table 1 compares the percentage changes of the new and incumbent policies with confidence intervals (CIs). Due to traffic limitations, LE results have much wider CIs than simulated metrics (we note they do overlap); the ability to produce tighter CIs are a significant advantage of simulation. Moreover, our simulation predictions are more closely aligned (informally, not statistically due to the wide LE CIs) with the real-world impact of the pCTR policy w.r.t. *actual post-launch metrics* obtained from 500,000 real users. Our simulation results also closely match the observed session length and artist-selection count distributions in post-launch logs, as seen in Figure 2.

Offline Optimization: We test whether our user models can be used to *improve* the RS policy using *purely offline* simulation with synthetic users. This would reduce both algorithm-development time and the user-cost of LEs. In its simplest form, we simulate three *new pCTR models*—these vary in their features and training data filters/ The best candidate, with a *simulated* CTR gain of +1.15%[−1.71%, 4.01%] w.r.t. the original pCTR model, was run in an LE with an observed CTR gain of +2.06%[0.20%, 3.92%]. It was subsequently deployed in production.

Trade-off λ	Selections change	Impressions change
0.001	0.07%[−3.35%, 3.49%]	0.75%[−0.08%, 1.59%]
0.01	0.20%[−3.16%, 3.56%]	−0.04%[−0.91%, 0.82%]
0.05	0.84%[−2.33%, 4.00%]	0.47%[−0.23%, 1.18%]
0.2	2.35%[−0.96%, 5.67%]	0.50%[−0.33%, 1.33%]
0.5	−0.48%[−3.69%, 2.72%]	−0.29%[−1.06%, 0.48%]
1	0.24%[−2.73%, 3.21%]	0.19%[−0.50%, 0.88%]
2	−0.78%[−4.03%, 2.46%]	0.75%[0.02%, 1.48%]
5	−1.67%[−5.22%, 1.88%]	0.18%[−0.78%, 1.15%]
10	−2.12%[−5.62%, 1.39%]	0.56%[−0.28%, 1.41%]

Table 2: Simulated Results (with 95% CIs) when Tuning the Trade-off between pCTR and Coverage.

	Sels. $\lambda = 0.001$	Sels. $\lambda = 0.05$	Sels. $\lambda = 0.2$	Ordering
LE 1	0.71 ± 3.35%	1.60 ± 1.91%	1.17 ± 3.81%	0.05>0.2>0.001
LE 2	1.62 ± 3.40%	0.54 ± 2.96%	0.52 ± 3.38%	0.001>0.05>0.2
LE 3	−2.11 ± 2.50%	0.07 ± 3.02%	0.61 ± 3.36%	0.2>0.05>0.001
Launch	−0.12 ± 1.88%	0.75 ± 1.22%	0.79 ± 1.98%	0.2>0.05>0.001
Simul.	0.07 ± 3.42%	0.84 ± 3.16%	2.35 ± 3.32%	0.2>0.05>0.001

Table 3: Selection Change (95% CIs) for 3 Policies, and Induced Policy Ordering: LEs vs. "Launch" vs. Simulation.

We next conducted *simulation-based experimentation* to optimize the OP PE policy offline. We study policies in which each query q is scored using $Score(q) = pCTR(q) + \lambda Coverage(q)$, where the first term is q's pCTR and the second, q's gain in artist space *coverage* (see Meshi et al. [15]). While a strong-performing production policy with $\lambda = 0.1$ was previously launched, we tested our ability to tune the trade-off parameter λ via simulation. Table 2 presents our offline simulation results on mobile traffic relative to the incumbent ($\lambda = 0.1$). It shows that $\lambda = 0.05$ and $\lambda = 0.2$ provide the greatest selection gains, with only modest impression increases, but with wide CIs. We ran LEs with these two policies ($\lambda = 0.05$, $\lambda = 0.2$), along with an under-performing candidate $\lambda = 0.001$, to test if simulation can correctly order policies. Specifically, we ran a large "Launch LE" with all three policies to measure their impact on artist selections. We then partitioned the data as if we did three smaller "true LEs" and assessed consistency with the launch data; we evaluate our simulation similarly.[4] Table 3 shows that none of the three small "true LEs" order the policies in the same way, and only one conforms to the "launch." By contrast, the simulation ordering predicts (informally) the actual ordering of the launch.

5 CONCLUSION

We demonstrated that simulation can play a key role in the evaluation and optimization of RS policies by reducing the need for costly LEs. We focused on new user onboarding in YouTube Music, showing that realistic user models can be learned from logged onboarding sessions and post-onboarding consumption. We described a simulation platform that allows integration of synthetic user models with full production infrastructure, providing suitable separation of live and simulated data, and showed the potential of our approach empirically. We hope this work will foster wider adoption of simulation for RS algorithm development and evaluation.

[4]The "LEs" and "Launch" metrics might be more correlated in our design than they would be in practice, since each LE is in fact a one-third sample of the Launch traffic.

6 PRESENTER'S BIO

Chih-Wei Hsu is a software engineer at Google Research. He is interested in user simulation, reinforcement learning, and machine learning in general. He has also worked on many real-world problems of recommender systems and ads. He received his Ph.D. in Computer Science from the University of Illinois Urbana-Champaign.

REFERENCES

[1] John Ahlgren, Kinga Bojarczuk, Sophia Drossopoulou, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Simon Mark Lucas, Erik Meijer, Steve Omohundro, Rubmary Rojas, Silvia Sapora, Jie M. Zhang, and Norm Zhou. 2021. Facebook's Cyber–Cyber and Cyber–Physical Digital Twins. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE-21)*. 1–9.

[2] Craig Boutilier. 2002. A POMDP Formulation of Preference Elicitation Problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*. Edmonton, 239–246.

[3] Urszula Chajewska, Daphne Koller, and Ronald Parr. 2000. Making Rational Decisions Using Adaptive Utility Elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*. Austin, TX, 363–369.

[4] Bassam H. Chaptini. 2005. *Use of Discrete Choice Models with Recommender Systems*. Ph. D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.

[5] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. 2018. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *12th ACM International Conference on Web Search and Data Mining (WSDM-19)*. Melbourne, Australia, 456–464.

[6] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the Twenty-eighth International Conference on Machine Learning (ICML-11)*. Bellevue, WA, 1097–1104.

[7] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *Proceedings of the Fourth International Conference on Learning Representations (ICLR-16)*. San Juan, Puerto Rico.

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[9] J. J. Hopfield. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences* 79, 8 (1982), 2554–2558.

[10] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SlateQ: A Tractable Decomposition for Reinforcement Learning with Recommendation Sets. In *Proceedings of the Twenty-eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. Macau, 2592–2599.

[11] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. (2019). `arXiv:1909.04847`.

[12] Q. Liu, S. Wu, D. Wang, Z. Li, and L. Wang. 2016. Context-Aware Sequential Recommendation. In *Proceedings of the IEEE International Conference on Data Mining (ICDM-16)*. Barcelona, 1053–1058.

[13] Jordan J. Louviere, David A. Hensher, and Joffre D. Swait. 2000. *Stated Choice Methods: Analysis and Application*. Cambridge University Press, Cambridge.

[14] Sean M. McNee, Shyong K. Lam, Joseph A. Konstan, and John Riedl. 2003. Interfaces for Eliciting New User Preferences in Recommender Systems. In *Proceedings of the 9th International Conference on User Modeling (UM-03)*. Johnstown, PA, 178–187.

[15] Ofer Meshi, Jon Feldman, Li Yang, Ben Scheetz, Yanli Cai, Mohammadhossein Bateni, Corbyn Salisbury, Vikram Aggarwal, and Craig Boutilier. 2023. Preference Elicitation for Music Recommendations. In *ICML 2023 Workshop The Many Facets of Preference-Based Learning*.

[16] Martin Mladenov, Elliot Creager, Kevin Swerksy, Omer Ben-Porat, Richard S. Zemel, and Craig Boutilier. 2020. Optimizing Long-term Social Welfare in Recommender Systems: A Constrained Matching Approach. In *Proceedings of the Thirty-seventh International Conference on Machine Learning (ICML-20)*. Vienna, 6987–6998.

[17] Martin Mladenov, Chih-wei Hsu, Vihan Jain, Eugene Ie, Chris Colby, Nic Mayoraz, Hubert Pham, Dustin Tran, Ivan Vendrov, and Craig Boutilier. 2021. RecSim NG: Toward Principled Uncertainty Modeling for Recommender Ecosystems. (2021). `arXiv:2103.08057`.

[18] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys-17)*. 130–137.

[19] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. (2018). `arXiv:1808.00720 [cs.IR]`.

[20] Guy Shani and Asela Gunawardana. 2011. Evaluating Recommendation Systems. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. Kantor (Eds.). Springer, Boston, 257–297.

[21] Paolo Viappiani and Craig Boutilier. 2010. Optimal Bayesian Recommendation Sets and Myopically Optimal Choice Query Sets. In *Advances in Neural Information Processing Systems 23 (NIPS)*. Vancouver, 2352–2360.

[22] Yueqi Wang, Yoni Halpern, Shuo Chang, Jingchen Feng, Elaine Ya Le, Longfei Li, Xujian Liang, Min-Cheng Huang, Shane Li, Alex Beutel, Yaping Zhang, and Shuchao Bi. 2023. Learning from Negative User Feedback and Measuring Responsiveness for Sequential Recommenders. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys-23)*. Singapore, 1049–1053.

[23] Siriu Yao, Yoni Halpern, Nithum Thain, Xuezhi Wang, Kang Lee, Flavien Prost, Ed H. Chi, Jilin Chen, and Alex Beutel. 2020. Measuring Recommender System Effects with Simulated Users. In *2nd Workshop on Fairness, Accountability, Transparency, Ethics and Society on the Web (FATES-20)*. Taipei.

[24] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining (WSDM-19)*. Melbourne, 582–590.

[25] Kesen Zhao, Shuchang Liu, Qingpeng Cai, Xiangyu Zhao, Ziru Liu, Dong Zheng, Peng Jiang, and Kun Gai. 2023. KuaiSim: A Comprehensive Simulator for Recommender Systems. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*. New Orleans.

[26] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys-18)*. Vancouver, 95–103.