Reinforcement Learning for Slate-based Recommender Systems: A Tractable Decomposition and Practical Methodology*

Eugene Ie[†,‡,1], Vihan Jain[‡,1], Jing Wang[‡,1], Sanmit Narvekar[§,2], Ritesh Agarwal[1], Rui Wu[1], Heng-Tze Cheng[1], Morgane Lustman[3], Vince Gatto[3], Paul Covington[3], Jim McFadden[3], Tushar Chandra[1], and Craig Boutilier[†,1]

[1]Google Research
[2]Department of Computer Science, University of Texas at Austin
[3]YouTube, LLC

June 3, 2019

Abstract

Most practical recommender systems focus on estimating immediate user engagement without considering the long-term effects of recommendations on user behavior. Reinforcement learning (RL) methods offer the potential to optimize recommendations for long-term user engagement. However, since users are often presented with slates of multiple items—which may have interacting effects on user choice—methods are required to deal with the combinatorics of the RL action space. In this work, we address the challenge of making slate-based recommendations to optimize long-term value using RL. Our contributions are three-fold. (i) We develop SLATEQ, a decomposition of value-based temporal-difference and Q-learning that renders RL tractable with slates. Under mild assumptions on user choice behavior, we show that the long-term value (LTV) of a slate can be decomposed into a tractable function of its component item-wise LTVs. (ii) We outline a methodology that leverages existing myopic learning-based recommenders to quickly develop a recommender that handles LTV. (iii) We demonstrate our methods in simulation, and validate the scalability of decomposed TD-learning using SLATEQ in live experiments on YouTube.

*Parts of this paper appeared in [Ie et al., 2019].

†Corresponding authors: {eugeneie,cboutilier}@google.com.

‡Authors Contributed Equally.

§Work done while at Google Research.

1 Introduction

Recommender systems have become ubiquitous, transforming user interactions with products, services and content in a wide variety of domains. In content recommendation, recommenders generally surface relevant and/or novel personalized content based on learned models of *user preferences* (e.g., as in collaborative filtering [Breese et al., 1998, Konstan et al., 1997, Srebro et al., 2004, Salakhutdinov and Mnih, 2007]) or predictive models of *user responses* to specific recommendations. Well-known applications of recommender systems include video recommendations on YouTube [Covington et al., 2016], movie recommendations on Netflix [Gomez-Uribe and Hunt, 2016] and playlist construction on Spotify [Jacobson et al., 2016]. It is increasingly common to train deep neural networks (DNNs) [van den Oord et al., 2013, Wang et al., 2015, Covington et al., 2016, Cheng et al., 2016] to predict user responses (e.g., click-through rates, content engagement, ratings, likes) to generate, score and serve candidate recommendations.

Practical recommender systems largely focus on *myopic* prediction—estimating a user's *immediate* response to a recommendation—without considering the long-term impact on subsequent user behavior. This can be limiting: modeling a recommendation's stochastic impact on the future affords opportunities to trade off user engagement in the near-term for longer-term benefit (e.g., by probing a user's interests, or improving satisfaction). As a result, research has increasingly turned to the sequential nature of user behavior using temporal models, such as hidden Markov models and recurrent neural networks [Rendle et al., 2010, Campos et al., 2014, He and McAuley, 2016, Sahoo et al., 2012, Tan et al., 2016, Wu et al., 2017], and long-term planning using *reinforcement learning (RL)* techniques (e.g., [Shani et al., 2005, Taghipour et al., 2007, Gauci et al., 2018]). However, the application of RL has largely been confined to restricted domains due to the complexities of putting such models into practice at scale.

In this work, we focus on the use of RL to maximize long-term value (LTV) of recommendations to the user, specifically, long-term user engagement. We address two key challenges facing the deployment of RL in practical recommender systems, the first algorithmic and the second methodological.

Our first contribution focuses on the algorithmic challenge of *slate recommendation* in RL. One challenge in many recommender systems is that, rather than a single item, multiple items are recommended to a user simultaneously; that is, users are presented with a *slate* of recommended items. This induces a RL problem with a large combinatorial action space, which in turn imposes significant demands on exploration, generalization and action optimization. Recent approaches to RL with such combinatorial actions [Sunehag et al., 2015, Metz et al., 2017] make inroads into this problem, but are unable to scale to problems of the size encountered in large, real-world recommender systems, in part because of their generality. In this work, we develop a new *slate decomposition* technique called SLATEQ that estimates the *long-term value (LTV)* of a slate of items by directly using the estimated LTV of the *individual items on the slate*. This decomposition exploits certain assumptions about the specifics of user choice behavior—i.e., the process by which user preferences dictate

selection and/or engagement with items on a slate—but these assumptions are minimal and, we argue below, very natural in many recommender settings.

More concretely, we first show how the SLATEQ decomposition can be incorporated into *temporal difference (TD)* learning algorithms, such as SARSA and Q-learning, so that LTVs can be learned at the level of individual items despite the fact that items are always presented to users in slates. This is critical for both generalization and exploration efficiency. We then turn to the optimization problem required to build slates that maximize LTV, a necessary component of policy improvement (e.g., in Q-learning) at training time and for selecting optimal slates at serving time. Despite the combinatorial (and fractional) nature of the underlying optimization problem, we show that it can be solved in polynomial-time by a two-step reduction to a linear program (LP). We also show that simple top-k and greedy approximations, while having no theoretical guarantees in this formulation, work well in practice.

Our second contribution is methodological. Despite the recent successes of RL afforded by deep Q-networks (DQNs) [Mnih et al., 2015, Silver et al., 2016], the deployment of RL in practical recommenders is hampered by the need to construct relevant state and action features for DQN models, and to train models that serve millions-to-billions of users. In this work, we develop a methodology that allows one to exploit existing myopic recommenders to: (a) accelerate RL model development; (b) reuse existing training infrastructure to a great degree; and (c) reuse the same serving infrastructure for scoring items based on their LTV. Specifically, we show how *temporal difference (TD)* learning can be built on top of existing myopic pipelines to allow the training and serving of DQNs.

Finally, we demonstrate our approach with both offline simulation experiments and online live experiments on the YouTube video recommendation system. We show that our techniques are scalable and offer significant improvements in user engagement over myopic recommendations. The live experiment also demonstrates how our methodology supports the relatively straightforward deployment of TD and RL methods that build on the learning infrastructure of extant myopic systems.

The remainder of the paper is organized as follows. In Section 2, we briefly discuss related work on the use of RL for recommender systems, choice modeling, and RL with combinatorial action spaces. We formulate the LTV slate recommendation problem as a Markov decision process (MDP) in Section 3 and briefly discuss standard value-based RL techniques, in particular, SARSA and Q-learning.

We introduce our SLATEQ decomposition in Section 4, discussing the assumptions under which the decomposition is valid, and how it supports effective TD-learning by allowing the long-term value (Q-value) of a slate to be decomposed into a function of its constituent item-level LTVs (Q-values). We pay special attention to the form of the *user choice model*, i.e., the random process by which a user's preferences determine the selection of an item from a slate. The decomposition affords *item-level exploration and generalization* for TD methods like SARSA and Q-learning, thus obviating the need to construct value or Q-functions explicitly over slates. For Q-learning itself to be feasible, we must also solve the combinatorial *slate optimization problem*—finding a slate with maximum LTV given the Q-values of individual

items. We address this problem in Section 5, showing that it can be solved effectively by first developing a fractional mixed-integer programming formulation for slate optimization, then deriving a reformulation and relaxation that allows the problem to be solved exactly as a linear program. We also describe two simple heuristics, *top-k* and *greedy* slate construction, that have no theoretical guarantees, but perform well in practice.

To evaluate these methods systematically, we introduce a recommender simulation environment, *RecSim*, in Section 6 that allows the straightforward configuration of an item collection (or vocabulary), a user (latent) state model and a user choice model. We describe specific instantiations of this environment suitable for slate recommendation, and in Section 7 we use these models in the empirical evaluation of our SLATEQ learning and optimization techniques.

The practical application of RL in the estimation of LTV in large-scale, practical recommender systems often requires integration of RL methods with production machine-learning training and serving infrastructure. In Section 8, we outline a general methodology by which RL methods like SLATEQ can be readily incorporated into the typical infrastructure used by many myopic recommender systems. We use this methodology to test the SLATEQ approach, specifically using SARSA to get one-step policy improvements, in a live experiment for recommendations on the YouTube homepage. We discuss the results of this experiment in Section 9.

2 Related Work

We briefly review select related work in recommender systems, choice modeling and combinatorial action optimization in RL.

Recommender Systems Recommender systems have typically relied on collaborative filtering (CF) techniques [Konstan et al., 1997, Breese et al., 1998]. These exploit user feedback on a subset of items (either explicit, e.g., ratings, or implicit, e.g., consumption) to directly estimate user preferences for unseen items. CF techniques include methods that explicitly cluster users and/or items, methods that embed users and items in a low-dimensional representation (e.g., LSA, probabilistic matrix factorization), or combinations of the two [Krestel et al., 2009, Moshfeghi et al., 2011].

Increasingly, recommender systems have moved beyond explicit preference prediction to capture more nuanced aspects of user behavior, for instance, how they respond to specific recommendations, such as pCTR (predicted click-through rate), degree of engagement (e.g., dwell/watch/listen time), ratings, social behavior (e.g., comments, sharing) and other behaviors of interest. DNNs now play a significant role in such approaches [van den Oord et al., 2013, Wang et al., 2015, Covington et al., 2016, Cheng et al., 2016] and often use CF-inspired embeddings of users and items as inputs to the DNN itself.

Sequence Models and RL in Recommender Systems Attempts to formulate recommendation as a RL problem have been relatively uncommon, though it has attracted more

attention recently. Early models included a MDP model for shopping recommendation [Shani et al., 2005] and Q-learning for page navigation [Taghipour et al., 2007], but were limited to very small-scale settings (100s of items, few thousands of users). More recently, biclustering has been combined with RL algorithms for recommendation [Choi et al., 2018], while Gauci et al. [2018] describe the use of RL in several applications at Facebook. Chen et al. [2018] also explored a novel off-policy policy-gradient approach that is very scalable, and was shown to be effective in a large-scale commercial recommender system. Their approach does not explicitly compute LTV improvements (as we do by developing Q-value models), nor does it model the slate effects that arise is practical recommendations.

Zhao et al. [2018] explicitly consider RL in slate-based recommendation systems, developing an actor-critic approach for recommending a page of items and tested using simulator trained on user logs. While similar in motivation to our approach, this method differs in several important dimensions: it makes no significant structural assumptions about user choice, using a CNN to model the spatial layout of items on a page, thus not handling the action-space combinatorics w.r.t. generalization, exploration, or optimization (but allowing additional flexibility in capturing user behavior). Finally, the focus of their method is online training and their evaluation with offline data is limited to item reranking.

Slate Recommendation and Choice Modeling Accounting for slates of items in recommender systems is quite common [Deshpande and Karypis, 2004, Boutilier et al., 2003, Viappiani and Boutilier, 2010, Le and Lauw, 2017], and the extension introduces interesting modeling questions (e.g., involving metrics such as diversity [Wilhelm et al., 2018]) and computational issues due to the combinatorics of slates themselves. Swaminathan et al. [2017] explored off-policy evaluation and optimization using inverse propensity scores in the context of slate interactions. Mehrotra et al. [2019] developed a hierarchical model for understanding user satisfaction in slate recommendation.

The construction of optimal recommendation slates generally depends on *user choice behavior*. Models of user choice from sets of items is studied under the banner of *choice modeling* in areas of econometrics, psychology, statistics, operations research and marketing and decision science [Luce, 1959, Louviere et al., 2000]. Probably the most common model of user choice is the *multinomial logit (MNL)* model and its extensions (e.g., the conditional logit model, the mixed logit model, etc.)—we refer to Louviere et al. [2000] for an overview.

For example, the conditional logit model assumes a set of user-item characteristics (e.g., feature vector) x_{ij} for user i and item j, and determines the (random) utility $u(x_{ij})$ of the item for the user. Typically, this model is linear so $u(x_{ij}) = \beta x_{ij}$, though we consider the use of DNN regressors to estimate these logits below. The probability of the user selecting j from a slate A of items is

$$P(j|A) = \frac{e^{u(x_{ij})}}{\sum_{\ell \in A} e^{u(x_{i\ell})}} \tag{1}$$

The choice model is justified under specific independence and extreme value assumptions [McFadden, 1974, Train, 2009]. Various forms of such models are used to model consumer choice and user behavior in wide ranging domains, together with specific methods for model

estimation, experiment design and optimization. Such models form the basis of optimization procedures in revenue management [Talluri and van Ryzin, 2004, Rusmevichientong and Topaloglu, 2012], product line design [Chen and Hausman, 2000, Schön, 2010], assortment optimization [Martínez-de Albéniz and Roels, 2011, Honhon et al., 2012] and a variety of other areas—we exploit connections with this work in Section 5 below.

The conditional logit model is an instance of a more general conditional choice format in which a user i selects item $j \in A$ with unnormalized probability $v(x_{ij})$, for some function v:

$$P(j|A) = \frac{v(x_{ij})}{\sum_{\ell \in A} v(x_{i\ell})}. \tag{2}$$

In the case of the conditional logit, $v(x_{ij}) = e^{\tau u(x_{ij})}$, but any arbitrary v can be used.

Within the ML community, including recommender systems and learning-to-rank, other choice models are used to explain user choice behavior. For example, *cascade models* [Joachims, 2002, Craswell et al., 2008, Kveton et al., 2015] have proven popular as a means of explaining user browsing behavior through (ordered) lists of recommendations, search results, etc., and is especially effective at capturing position bias. The standard cascade model assumes that a user i has some affinity (e.g., perceived utility) u_{ij_k} for any item j_k; sequentially scans a list of items $A = (j_1, j_2, \ldots, j_K)$ in order; and will select (e.g., click) an item with probability $\phi(u_{ij_k})$ for some non-decreasing function ϕ. If an item is selected when inspected, no items following will be inspected/selected; and if the last item is inspected but not selected, then no selection is made. Thus the probability of j_k being selected is:

$$P(j_k|A) = \prod_{\ell<k} (1 - \phi(u_{ij_\ell}))\phi(u_{ij_k}). \tag{3}$$

Various mechanisms for model estimation, optimization and exploration have been proposed for the basic cascade model and its variations. Recently, DNN and sequence models have been developed for explaining user choice behavior in a more general, non-parametric fashion [Ai et al., 2018, Bello et al., 2018]. As one example, Jiang et al. [2019] proposed a slate-generation model using conditional variational autoencoders to model the distribution of slates conditioned on user response, but the scalability requires the use of a pre-trained item embedding in large domains of the type we consider. However, the CVAE model does offer considerably flexibility in capturing item interactions, position bias, and other slate effects that might impact user response behavior.

RL with Combinatorial Action Spaces Designing tractable RL approaches for *combinatorial actions*—of which slates recommendations are an example—is itself quite challenging. Some recent work in recommender systems considered slate-based recommendations (see, e.g., discussion of Zhao et al. [2018] above, though they do not directly address the combinatorics), though most is more general. *Sequential DQN* [Metz et al., 2017] decomposes k-dimensional actions into a sequence of atomic actions, inserting fictitious states between them so a standard RL method can plan a trajectory giving the optimal action configuration.

While demonstrated to be useful in some circumstances, the approach trades off the exponential size of the action space with a corresponding exponential increase in the size of the state space (with fictitious states corresponding to possible sequences of sub-actions).

Sunehag et al. [2015] proposed *Slate MDPs* which considers slates of *primitive actions*, using DQN to learn the value of item slates, and a greedy procedure to construct slates. In fact, they develop three DQN methods for the problem, two of which manage the combinatorics of slates by assuming the primitive actions can be executed in isolation. In our setting, this amounts to the unrealistic assumption that we could "force" a user to consume a specific item (rather than present them with a slate, from which no item might be consumed). Their third approach, *Generic Full Slate*, makes no such assumption, but maintains an explicit Q-function over slates of items. This means it fails to address the exploration and generalization problems, and while the greedy optimization (action selection) method used is tractable, it comes with no guarantees of optimality.

3 An MDP Model for Slate Recommendation

In this section, we develop a *Markov decision process (MDP)* model for content recommendation with *slates*. We consider a setting in which a recommender system is charged with presenting a slate to a user, from which the user selects zero or more items for consumption (e.g., listening to selected music tracks, reading content, watching video content). Once items are consumed, the user can return for additional slate recommendations or terminate the session. The user's response to a consumed item may have multiple dimensions. These may include the immediate degree of engagement with the item (e.g., consumption time); ratings feedback or comments; sharing behavior; subsequent engagement with the content provider beyond the recommender system's direct control. In this work, we use *degree of engagement* abstractly as the reward without loss of generality, since it can encompass a variety of metrics, or their combinations.

We focus on *session optimization* to make the discussion concrete, but our decomposition applies equally well to any long-term horizon.[1] Session optimization with slates can be modeled as a MDP with states $\mathcal{S}$, actions $\mathcal{A}$, reward function R and transition kernel P, with discount factor $0 \leq \gamma \leq 1$.

The states $\mathcal{S}$ typically reflect *user state*. This includes relatively static user features such as demographics, declared interests, and other user attributes, as well as more dynamic user features, such as user context (e.g., time of day). In particular, summaries of relevant user history and past behavior play a key role, such as past recommendations made to the user; past user responses, such as recommendations accepted or passed on, the specific items consumed, and degree of user engagement with those items. The summarization of history is often domain specific (see our discussion of methodology in Section 8) and can be viewed as a means of capturing certain aspects of user latent state in a partially observable MDP. The

[1] Dealing with very extended horizons, such as *lifetime value* [Theocharous et al., 2015, Hallak et al., 2017], is often problematic for any RL method; but such issues are independent of the slate formulation and decomposition we propose.

state may also reflect certain general (user-independent) environment variables. We develop our model assuming a finite state space for ease of exposition, though our experiments and our methodology admit both countably infinite and continuous state features.

The action space $\mathcal{A}$ is simply the set of all possible recommendation slates. We assume a fixed catalog of recommendable items $\mathcal{I}$, so actions are the subsets $A \subseteq \mathcal{I}$ such that $|A| = k$, where k is the slate size. We assume that each item $a \in \mathcal{I}$ and each slate A is recommendable at each state s for ease of exposition. However, our methods apply readily when certain items cannot be recommended at particular states by specifying $\mathcal{I}_s$ for each $s \in \mathcal{S}$ and restricting $\mathcal{A}_s$ to subsets of $\mathcal{I}_s$. If additional constraints are placed on slates so that $\mathcal{A}_s$ is a *strict* subset of the slates defined over $\mathcal{I}_s$, these can be incorporated into the slate optimization problem at both training and serving time.[2] We do not account for positional bias or ordering effects within the slate in this work, though such effects can be incorporated into the choice model (see below).

To account for the fact that a user may select no item from a slate, we assume that every slate includes a $(k+1)$st *null item*, denoted $\perp$. This is standard in most choice modeling work and makes it straightforward to specify all user behavior as if induced by a choice from the slate.

Transition probability $P(s'|s, A)$ reflects the probability that the user transitions to state s' when action A is taken at user state s. This generally reflects uncertainty in both user response and the future contextual or environmental state. One of the most critical points of uncertainty pertains the probability with which a user will consume a particular recommended item $a \in A$ from the slate. As such, choice models play a critical role in evaluating the quality of a slate as we detail in the next section.

Finally, the reward $R(s, A)$ captures the expected reward of a slate, which measures the expected degree of user engagement with items on the slate. Naturally, this expectation must account for the uncertainty in user response.

Our aim is to find optimal slate recommendation as a function of the state. A *(stationary, deterministic) policy* $\pi : \mathcal{S} \to \mathcal{A}$ dictates the action $\pi(s)$ to be taken (i.e., slate to recommend) at any state s. The *value function* V^π of a fixed policy π is:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s'). \tag{4}$$

The corresponding action value, or *Q-function*, reflects the value of taking an action a at state s and then acting according to π:

$$Q^\pi(s, A) = R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^\pi(s'). \tag{5}$$

The *optimal policy* π^* maximizes expected value $V(s)$ uniformly over $\mathcal{S}$, and its value—

[2] We briefly describe where relevant adjustments are needed in our algorithms when we present them. We also note that our methods work equally well when the feasible set of slates $\mathcal{A}_s$ is stochastic (but stationary) as in [Boutilier et al., 2018].

the *optimal value function* V^*—is given by the fixed point of the Bellman equation:

$$V^*(s) = \max_{A \in \mathcal{A}} R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^*(s'). \tag{6}$$

The optimal Q-function is defined similarly:

$$Q^*(s, A) = R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^*(s'). \tag{7}$$

The optimal policy satisfies $\pi^*(s) = \arg\max_{A \in \mathcal{A}} Q^*(s, A)$.

When transition and reward models are both provided, optimal policies and value functions can be computed using a variety of methods [Puterman, 1994], though generally these require approximation in large state/action problems [Bertsekas and Tsitsiklis, 1996]. With sampled data, RL methods such as TD-learning [Sutton, 1988], *SARSA* [Rummery and Niranjan, 1994, Sutton, 1996] and *Q-learning* [Watkins and Dayan, 1992] can be used (see [Sutton and Barto, 1998] for an overview). Assume training data of the form (s, A, r, s', A') representing observed transitions and rewards generated by some policy π. The Q-function Q^π can be estimated using SARSA updates of the form:

$$Q^{(t)}(s, A) \leftarrow \alpha^{(t)}[r + \gamma Q^{(t-1)}(s', A')] + (1 - \alpha^{(t)}) Q^{(t-1)}(s, A), \tag{8}$$

where $Q^{(t)}$ represents the tth iterative estimate of Q^π and α is the learning rate. SARSA, Eq. (8), is *on-policy* and estimates the value of the data-generating policy π. However, if the policy has sufficient exploration or other forms of stochasticity (as is common in large recommender systems), acting greedily w.r.t. Q^π, and using the data so-generated to train a new Q-function, will implement a policy improvement step [Sutton and Barto, 1998]. With repetition—i.e., if the updated Q^π is used to make recommendations (with some exploration), from which new training data is generated—the process will converge to the optimal Q-function. Note that acting greedily w.r.t. Q^π requires the ability to compute optimal slates at serving time. In what follows, we use the term SARSA to refer to the (on-policy) estimation of the Q-function $Q^\pi(s, a)$ of a *fixed* policy π, i.e., the TD-prediction problem on state-action pairs.[3]

The optimal Q-function Q^* can be estimated directly in a related fashion:

$$Q^{(t)}(s, A) \leftarrow \alpha^{(t)}[r + \max_{A'} \gamma Q^{(t-1)}(s', A')] + (1 - \alpha^{(t)}) Q^{(t-1)}(s, A). \tag{9}$$

where $Q^{(t)}$ represents the tth iterative estimate of Q^*. Q-learning, Eq. (9), is *off-policy* and directly estimates the optimal Q-function (again, assuming suitable randomness in the data-generating policy π). Unlike SARSA, Q-learning requires that one compute optimal slates A' at training time, not just at serving time.

[3] SARSA is often used to refer to the on-policy *control* method that includes making policy improvement steps. We use it simply to refer to the TD-method based on SARSA updates as in Eq. (8).

4 SLATEQ: Slate Decomposition for RL

One key challenge in the formulation above is the combinatorial nature of the action space, consisting of all $\binom{|\mathcal{I}|}{k} \cdot k!$ ordered k-sets over $\mathcal{I}$. This poses three key difficulties for RL methods. First, the sheer size of the action space makes sufficient *exploration* impractical. It will generally be impossible to execute all slates even once at any particular state, let alone satisfy the sample complexity requirements of TD-methods. Second, *generalization* of Q-values across slates is challenging without some compressed representation. While a slate could be represented as the collection of features of its constituent items, this imposes greater demands on sample complexity; we may further desire greater generalization capabilities. Third, we must solve the combinatorial optimization problem of finding a slate with maximum Q-value—this is a fundamental part of Q-learning and a necessary component in any form of policy improvement. Without significant structural assumptions or approximations, such optimization cannot meet the real-time latency requirements of production recommender systems (often on the order of tens of milliseconds).

In this section, we develop *SlateQ*, a model that allows the Q-value of a slate to be *decomposed into a combination of the item-wise Q-values of its constituent items*. This decomposition exposes precisely the type of structure needed to allow effective exploration, generalization and optimization. We focus on the SLATEQ decomposition in this section—the decomposition itself immediately resolves the exploration and generalization concerns. We defer discussion of the optimization question to Section 5.

Our approach depends to some extent on the nature of the user choice model, but critically on the interaction it has with subsequent user behavior, specifically, how it influences both expected engagement (i.e., reward) and user latent state (i.e., state transition probabilities). We require two assumptions to derive the SLATEQ decomposition.

- **Single choice (SC):** A user consumes a *single* item from each slate (which may be the *null item* $\perp$).
- **Reward/transition dependence on selection (RTDS):** The realized reward (user engagement) $R(s, A)$ depends (perhaps stochastically) *only on the item* $i \in A$ *consumed by the user* (which may also be the *null item* $\perp$). Similarly, the state transition $P(s'|s, A)$ depends only on the consumed $i \in A$.

Assumption **SC** implies that the user selection of a subset $B \subseteq A$ from slate A has $P(B|s, A) > 0$ only if $|B| = 1$. While potentially limiting in some settings, in our application (see Section 9), users can consume only one content item at a time. Returning to the slate for a second item is modeled and logged as a separate event, with the user making a selection in a new state that reflects engagement with the previously selected item. As such, **SC** is valid in our setting.[4] Letting $R(s, A, i)$ denote the reward when a user in state s,

[4]Domains in which the user can select multiple items without first engaging with them (i.e., without induced some change in state) would be more accurately modeled by allowing multiple selection. Our SLATEQ model can be extended to incorporate a simple correction term to accurately model user selection of multiple items by assuming conditional independence of item-choice probabilities given A.

presented with slate A, selects item $i \in A$, and $P(s'|s, A, i)$ the corresponding probability of a transition to s', the **SC** assumption allows us to express immediate rewards and state transitions as follows:

$$R(s, A) = \sum_{i \in A} P(i|s, A) R(s, A, i), \tag{10}$$

$$P(s'|s, A) = \sum_{i \in A} P(i|s, A) P(s'|s, A, i). \tag{11}$$

The **RTDS** assumption is also realistic in many recommender systems, especially with respect to immediate reward. It is typically the case that a user's engagement with a selected item is not influenced to a great degree by the options in the slate that were not selected. The transition assumption also holds in recommender systems where direct user interaction with items drives user utility, overall satisfaction, new interests, etc., and hence is the primary determinant of the user's underlying latent state. Of course, in some recommender domains, unconsumed items in the slate (say, impressions of content descriptions, thumbnails, clips, etc) may themselves create, say, future curiosity, which should be reflected by changes in the user's latent state. But even in such cases **RTDS** may be treated as a reasonable simplifying assumption, especially where such impressions have significantly less impact on the user than consumed items themselves. The **RTDS** assumption can be stated as:

$$R(s, A, i) = R(s, A', i) = R(s, i), \quad \forall A, A' \text{ containing } i, \tag{12}$$

$$P(s'|s, A, i) = P(s'|s, A', i) = P(s'|s, i), \quad \forall A, A' \text{ containing } i. \tag{13}$$

Our decomposition of (on-policy) Q-functions for a fixed data-generating policy π relies on an *item-wise auxiliary function* $\overline{Q}^{\pi}(s, i)$, which represents the LTV of a user consuming an item i, i.e., the LTV of i conditional on it being clicked. Under **RTDS**, this function is independent of the slate A from which i was selected. We define:

$$\overline{Q}^{\pi}(s, i) = R(s, i) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, i) V^{\pi}(s'). \tag{14}$$

Incorporating the **SC** assumption, we immediately have:

Proposition 1. $Q^{\pi}(s, A) = \sum_{i \in A} P(i|s, A) \overline{Q}^{\pi}(s, i)$.

Proof. This holds since:

$$Q^{\pi}(s, A) = R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^{\pi}(s') \tag{15}$$

$$= \sum_{i \in A} P(i|s, A) R(s, i) + \gamma \sum_{i \in A} P(i|s, A) \sum_{s' \in \mathcal{S}} P(s'|s, i) V^{\pi}(s') \tag{16}$$

$$= \sum_{i \in A} P(i|s, A) [R(s, i) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, i) V^{\pi}(s')] \tag{17}$$

$$= \sum_{i \in A} P(i|s, A) \overline{Q}^{\pi}(s, i). \tag{18}$$

Here Eq. (16) follows immediately from **SC** and **RTDS** (see Eqs. (10, 11, 12, 13)) and Eq. (18) follows from the definition of $\overline{Q}^{\pi}$ (see Eq. 14). □

This simple result gives a *complete decomposition* of slate Q-values into Q-values for individual items. Thus, the combinatorial challenges disappear if we can learn $\overline{Q}^{\pi}(s, i)$ using TD methods. Notice also that the decomposition exploits the existence of a known choice function. But apart from knowing it (and using it our Q-updates that follow), we make no particular assumptions about the choice model apart from **SC**. We note that learning choice models from user selection data is generally quite routine. We discuss specific choice functions in the next section and how they can be exploited in optimization.

TD-learning of the function $\overline{Q}^{\pi}$ can be realized using a very simple Q-update rule. Given a consumed item i at s with observed reward r, a transition to s', and selection of slate $\pi(s') = A'$, we update $\overline{Q}^{\pi}$ as follows:

$$\overline{Q}^{\pi}(s, i) \leftarrow \alpha(r + \gamma \sum_{j \in A'} P(j|s', A')\overline{Q}^{\pi}(s', j)) + (1 - \alpha)\overline{Q}^{\pi}(s, i). \tag{19}$$

The soundness of this update follows immediately from Eq. 14.

Our decomposed SLATEQ update facilitates more compact Q-value models, *using items as action inputs rather than slates*. This in turn allows for greater generalization and data efficiency. Critically, while SLATEQ learns item-level Q-values, it can be shown to converge to the correct *slate Q-values* under standard assumptions:

Proposition 2. *Under standard assumptions on learning rate schedules and state-action exploration [Sutton, 1988, Dayan, 1992, Sutton and Barto, 1998], and the assumptions on user choice probabilities, state transitions, and rewards stated in the text above,* SLATEQ*—using update (19) and definition of slate value (18)—will converge to the true slate Q-function* $Q^{\pi}(s, A)$ *and support greedy policy improvement of* π.

Proof. (Brief sketch.) Standard proofs of convergence of TD(0), applied to the state-action Q-function Q^{π} apply directly, with the exception of the introduction of the direct *expectation* over user choices, i.e., $\sum_{j \in A'} P(j|s', A')$, rather than the use of sampled choices.[5] However, it is straightforward to show that incorporating the explicit expectation does not impact the convergence of TD(0) (see, for example, the analysis of *expected SARSA* [Van Seijen et al., 2009]). There is some additional impact of user choice on exploration policies as well—if the choice model is such that some item j has choice probability $P(j|s, A) = 0$ for *any slates* A *with* $\pi(s) > 0$ in some state s, we will not experience user selection of item j at state s under π (for value prediction of V^{π} this is not problematic, but it is for learning a Q^{π}). Thus the exploration policy must account for the choice model, either by sampling all slates at each state (which is very inefficient), or by configuring exploratory slates that ensure each item j is sampled sufficiently often. For most common choice models (see discussion below), every item has non-zero probability of selection, in which case, standard action-level exploration conditions apply. □

[5] We note that sampled choice could also be used in the full on-policy setting, but is problematic for optimization/action maximization as we discuss below.

Notice that update (19) requires the use of a known choice model. Such choice models are quite commonly learned in ML-based recommender systems, as we discuss further below in Section 8. The introduction of this expectation—rather than relying on sampled user choices—can be viewed as reducing variance in the estimates much like *expected SARSA*, as discussed by Sutton and Barto [1998] and analyzed formally by Van Seijen et al. [2009]. Furthermore, it is straightforward to show that the standard SARSA(0) algorithm (with policy improvement steps) will converge to the optimal Q-function, subject to the considerations mentioned above, using standard techniques [Singh et al., 2000, Van Seijen et al., 2009].

The decomposition can be applied to Q-learning of the optimal Q-function as well, requiring only a straightforward modification of Eq. (14) to obtain $\overline{Q}(s,i)$, the optimal (off-policy) conditional-on-click item-wise Q-function, specifically, replacing $V^{\pi}(s')$ with $V^{*}(s')$ (the proof is analogous to that of Proposition 1):

Proposition 3. $Q(s,A) = \sum_{i \in A} P(i|s,A)\overline{Q}(s,i)$.

Likewise, extending the decomposed update Eq. (19) to full Q-learning requires only that we introduce the usual maximization:

$$\overline{Q}(s,i) \leftarrow \alpha(r + \gamma \max_{A' \in \mathcal{A}} \sum_{j \in A'} P(j|s',A')\overline{Q}(s',j)) + (1-\alpha)\overline{Q}(s,i). \tag{20}$$

As above, it is not hard to show that Q-learning using this update will converge, using standard techniques [Watkins and Dayan, 1992, Van Seijen et al., 2009] and with similar considerations to those discussed in the proof sketch of Proposition 2:

Proposition 4. *Under standard assumptions on learning rate schedules and sufficient exploration [Sutton and Barto, 1998], and the assumptions on user choice probabilities, state transitions, and rewards stated in the text above,* SLATEQ*—using update (20) and definition of slate value in Proposition 3—will converge to the optimal slate Q-function* $Q^{*}(s,A)$.

The decomposition of both the policy-based and optimal Q-functions above accomplishes two of our three desiderata: it circumvents the natural combinatorics of both exploration and generalization. But we still face the combinatorics of action maximization: the *LTV slate optimization problem* is the combinatorial optimization problem of selecting the optimal slate from $\mathcal{A}$, the space of all $\binom{|\mathcal{I}|}{k}k!$ possible (ordered) k-sets over $\mathcal{I}$. This is required during training with Q-learning (Eq. (9)) and when engaging in policy improvement using SARSA. One also needs to solve the slate optimization problem at serving time when executing the induced greedy policy (i.e., presenting slates with maximal LTV to users given a learned Q-function). In the next section, we show that exact optimization is tractable and also develop several heuristic approaches to tackling this problem.

5 Slate Optimization with Q-values

We address the LTV slate optimization in this section. We develop an exact linear programming formulation of the problem in Section 5.1 using (a generalization of) the conditional

logit model. In Section 5.2, we describe two computationally simpler heuristics for the problem, the top-k and greedy algorithms.

5.1 Exact Optimization

We formulate the *LTV slate optimization problem* as follows:

$$\max_{\substack{A \subset \mathcal{I} \\ |A|=k}} \sum_{i \in A} P(i|s, A)\overline{Q}(s, i). \tag{21}$$

Intuitively, a user makes her choice from the slate based on the perceived properties (e.g., attractiveness, quality, topic, utility) of the constituent items. In the LTV slate optimization problem, we value the selection of an item from the slate based on its LTV or $\overline{Q}$-value, rather than its immediate appeal to the user. As discussed above, we assume access to the choice model $P(i|s, A)$, since models (e.g., pCTR models) predicting user selection from a slate are commonly used in myopic recommenders. Of course, the computational solution of the slate optimization problem depends on the form of the choice model. We discuss the use of the *conditional logit model (CLM)* in SLATEQ (and the more general format, Eq. (2)) in this subsection.

When using the conditional logit model (see Eq. 1), the LTV slate optimization problem is analogous in a formal sense to, *assortment optimization* or *product line design* [Chen and Hausman, 2000, Schön, 2010, Rusmevichientong and Topaloglu, 2012], in which a retailer designs or stocks a set of k products whose expected revenue or profit is maximized assuming that consumers select products based on their appeal (and not their value to the retailer).[6]

Our optimization formulation is suited to any general conditional choice model of the form Eq. (2) (of which the conditional logit is an instance).[7] We assume a user in state s selects item $i \in A$ with unnormalized probability $v(s, i)$, for some function v. In the case of the conditional logit, $v(s, i) = e^{\tau u(s,i)}$. We can express the optimization Eq. (21) w.r.t. such a v as a fractional mixed-integer program (MIP), with binary variables $x_i \in \{0, 1\}$ for each

[6]Naturally, there are more complex variants of assortment optimization, including the choice of price, inclusion of fixed production or inventory costs, etc. There are other conceptual differences with our model as well. While not a formal requirement, LTV of an item in our setting reflects user engagement, hence reflects some form of user satisfaction as opposed to direct value to the recommender. In addition, many assortment optimization models are designed for consumer populations, hence choice probabilities are often reflective of diversity in the population (though random selection by individual consumers is sometimes considered as well; by contrast, in the recommender setting, choice probabilities are usually dependent on the features of individual users and typically reflect the recommender's uncertainty about a user's immediate intent or context.

[7]We note that since the ordering of items within a slate does not impact choice probabilities in this model, the action (or slate) space consists of the $\binom{\mathcal{I}}{k}$ *unordered* k-sets in this case.

item $i \in \mathcal{I}$ indicating whether i occurs in slate A:

$$\max \sum_{i \in \mathcal{I}} \frac{x_i v(s,i) \overline{Q}(s,i)}{v(s,\perp) + \sum_j x_j v(s,j)} \tag{22}$$

$$\text{s.t.} \sum_{i \in \mathcal{I}} x_i = k; \quad x_i \in \{0,1\},\ \forall i \in \mathcal{I}. \tag{23}$$

This is a variant of a classic product-line (or assortment) optimization problem [Chen and Hausman, 2000, Schön, 2010]. Our problem is somewhat simpler since there are no fixed resource costs or per-item costs.

Chen and Hausman [2000] show that that the binary indicators in this MIP can be relaxed to obtain the following fractional linear program (LP):

$$\max \sum_{i \in \mathcal{I}} \frac{x_i v(s,i) \overline{Q}(s,i)}{v(s,\perp) + \sum_j x_j v(s,j)} \tag{24}$$

$$\text{s.t.} \sum_{i \in \mathcal{I}} x_i = k; \quad 0 \le x_i \le 1,\ \forall i \in \mathcal{I}. \tag{25}$$

The constraint matrix in this relaxed problem is totally unimodular, so the optimal (vertex) solution is integral and standard non-linear optimization methods can be used. However, since it is a fractional LP, it is directly amenable to the Charnes-Cooper 1962 transformation and can be recast directly as a (non-fractional) LP. To do so, we introduce an additional variable t that implicitly represents the *inverse* choice weight of the selected items $t = (v(s,\perp) + \sum_j x_j v(s,j))^{-1}$, and auxiliary variables y_i that represent the products $x_i \cdot (v(s,\perp) + \sum_j x_j v(s,j))^{-1}$, giving the following LP:

$$\max \sum_i y_i v(s,i) \overline{Q}(s,i) \tag{26}$$

$$\text{s.t. } tv(s,\perp) + \sum_i y_i v(s,i) = 1 \tag{27}$$

$$t \ge 0; \quad \sum_i y_i \le kt. \tag{28}$$

The optimal solution $(\mathbf{y}^*, t^*)$ to this LP yields the optimal x_i assignment in the fractional LP Eq. (24) via $x_i = y_i^*/t^*$, which in turn gives the optimal slate in the original fractional MIP Eq. (22)—just add any item to the slate where $y_i^* > 0$. This formulation applies equally well to the MNL model, or related random utility models. The slate optimization problem is now immediately proven to be polynomial-time solvable.

Observation 5. *The LTV slate optimization problem Eq. 21, under the general conditional choice model (including the conditional logit model), is solvable in polynomial-time in the number of items $|\mathcal{I}|$ (assuming a fixed slate size k).*

Thus full Q-learning with slates using the SLATEQ decomposition imposes at most a small polynomial-time overhead relative to item-wise Q-learning despite its combinatorial nature. We also note that many production recommender systems limit the set of items to be ranked using a separate retrieval policy, so the set of items to consider in the LP is usually much smaller than the complete item set. We discuss this further below in Section 8.

5.2 Top-k and Greedy Optimization

While the exact maximization of slates under the conditional choice model can be accomplished in polynomial-time using $\overline{Q}$ and the item-score function v, we may wish to avoid solving an LP at serving time. A natural heuristic for constructing a slate is to simply add the k items with the highest score. In our *top-k optimization* procedure, we insert items into the slate in decreasing order of the product $v(s,i)\overline{Q}(s,i)$.[8] This incurs only a $O(\log(\mathcal{I}))$ overhead relative to the $O(\mathcal{I})$ time required for maximization for item-wise Q-learning.

One problem with top-k optimization is the fact that, when considering the item to add to the Lth slot (for $1 < L \leq k$), item scores are not updated to reflect the previous $L-1$ items already added to the slate. *Greedy optimization*, instead of scoring each item *ab initio*, updates item scores with respect to the current partial slate. Specifically, given $A' = \{i_{(1)}, \ldots i_{(L-1)}\}$ of size $L-1 < k$, the Lth item added is that with maximum marginal contribution:

$$\arg\max_{i \notin A'} \frac{v(s,i)\overline{Q}(s,i) + \sum_{\ell<L} v(s,i_{(\ell)})\overline{Q}(s,i_{(\ell)})}{v(s,i) + v(s,\perp) + \sum_{\ell<L} v(s,i_{(\ell)})}.$$

We compare top-k and greedy optimizations with the LP solution in our offline simulation experiments below.

Under the general conditional choice model (including for the conditional logit model), neither top-k nor greedy optimization will find the optimal solution as following counterexample illustrates:

Item	Score ($v(s,i)$)	Q-value
$\perp$	1	0
a	2	0.8
b_1, b_2	1	1

The null item is always on the slate. Items b_1, b_2 are identical w.r.t. their behavior. We have $V(\{a\}) = 1.6/3$, greater than $V(\{b_i\}) = 1/2$. Both top-k and greedy will place a on the slate first. However, $V(\{a, b_i\}) = 2.6/4$, whereas the optimal slate $\{b_1, b_2\}$ is valued at $2/3$. So for slate size $k = 2$, neither top-k nor greedy find the optimal slate.

While one might hope that the greedy algorithm provides some approximation guarantee, the set function is not submodular, which prevents standard analyses (e.g., [Nemhauser et al., 1978, Feige, 1998, Buchbinder et al., 2014]) from being used. The following example illustrates this.

[8] Top-k slate construction is quite common in slate-based myopic recommenders. It has recently been used in LTV optimization as well [Chen et al., 2018].

Item	Score ($v(s,i)$)	Q-value
$\perp$	1	10
a	1	10
b	2	ε

We have expected values of the following item sets: $V(\emptyset) = 10; V(\{a\}) = 10; V(\{b\}) = (10+\varepsilon)/3; V(\{a,b\}) = 5 + \varepsilon/2$. The fact that $V(\{a\}) - V(\emptyset) < V(\{a,b\}) - V(\{b\})$ demonstrates lack of submodularity (the set function is also not monotone).[9]

While we have no current performance guarantees for greedy and top-k, it's not hard to show that top-k can perform arbitrarily poorly.

Observation 6. *The approximation ratio of top-k optimization for slate construction is unbounded.*

The following example demonstrates this.

Item	Score ($v(s,i)$)	Q-value
$\perp$	ε	0
a	ε	1
b	1	ε

Suppose we have $k = 1$. Top-k scores item b higher than a, creating the slate with value $V(\{b\}) = \varepsilon/(1+\varepsilon)$, while the optimal slate has value $V(\{a\}) = 1/2$.

5.3 Algorithm Variants

With a variety of slate optimization methods at our disposal, many variations of RL algorithms exist depending on the optimization method used during training and serving. Given a trained SLATEQ model, we can apply that model to *serve* users using either top-k, greedy or the LP-based optimal method to generate recommended slates. Below we use the designations TS, GS, or OS to denote these serving protocols, respectively. These designations apply equally to (off-policy) Q-learned models, (on-policy) SARSA models, and even (non-RL) *myopic* models.[10]

During Q-learning, slate optimization is also required at *training time* to compute the maximum successor state Q-value (Eq. 20). This can also use either of the three optimization methods, which we designate by TT, GT, and OT for top-k, greedy and optimal LP training, respectively. This designation is not applicable when training a myopic model or SARSA (since SARSA is trained only on-policy). This gives us the following collection of algorithms. For Q-learning, we have:

[9] It is worth observing that without our exact cardinality constraint (sets must have size k), the optimal set under MNL can be computed in a greedy fashion [Talluri and van Ryzin, 2004] (the analysis also applies to the conditional logit model).

[10] A myopic model is equivalent to a Q-learned model with $\gamma = 0$.

		Serving		
		Top-k	Greedy	LP (Opt)
Training	Top-k	QL-TT-TS	QL-TT-GS	QL-TT-OS
	Greedy	QL-GT-TS	QL-GT-GS	QL-GT-OS
	LP (Opt)	QL-OT-TS	QL-OT-GS	QL-OT-OS

For SARSA and Myopic recommenders, we have:

Serving	SARSA	Myopic
Top-k	SARSA-TS	MYOP-TS
Greedy	SARSA-GS	MYOP-GS
LP (Opt)	SARSA-OS	MYOP-OS

In our experiments below we also consider two other baselines: Random, which recommends random slates from the feasible set; and *full-slate Q-learning (FSQ)*, which is a standard, non-decomposed Q-learning method that treats each slate *atomically (i.e., holistically)* as a single action. The latter is a useful baseline to test whether the SLATEQ decomposition provides leverage for generalization and exploration.

5.4 Approaches for Other Choice Models

The SLATEQ decomposition works with any choice model that satisfies the assumptions **SC** and **RTDS**, though the form of the slate optimization problem depends crucially on the choice model. To illustrate, we consider the *cascade choice model* outlined in Section 2 (see, e.g., Eq. (3)). Notice that the cascade model, unlike the general conditional choice model, has position-dependent effects (i.e., reordering of items in the slate changes selection probabilities and the expected LTV of the slate). However, it is not hard to show that the cascade model exhibits a form of "ordered submodularity" if we assume that the LTV or conditional Q-value of not selecting from the slate is no greater that the Q-value of selecting any item on the slate, i.e., if $Q(s, i) \geq Q(s, \perp)$ for all $i \in \mathcal{I}$.[11] Specificially, the value of the marginal increase in value induced by adding item $i_{\ell+1}$ to the (ordered) partial slate $(i_1, i_2, \ldots i_\ell)$ is no greater than the increase in value of adding $i_{\ell+1}$ to a prefix of that slate $(i_1, i_2, \ldots i_j)$ for any $j < \ell$. Thus top-k optimization can be used to support training and serving of the SLATEQ approach under the cascade model.[12]

While the general conditional choice model is order-independent, in practice, it may be the case that users incorporate some elements of a cascade-like model into the conditional choice model. For example, users may devote a *random* amount of time t or effort to inspect a slate of k recommended items, compare the top $k' \leq k$ items, where $k' = F(t)$ is some function of the available time, and select (perhaps noisily) the most preferred item from among those inspected. This model would be a reasonable approximation of user behavior

[11] The statements to follow hold under the weaker condition that, for all states s, there are at least k items $i_1^s, \ldots i_k^s$ such that $Q(s, i_j^s) \geq Q(s, \perp), \forall j \leq k$ (where k is the slate size).

[12] It is also not hard to show that top-k is not optimal for the cascade model.

in the case of recommenders that involve scrolling interfaces for example. In such a case, we end up with a distribution over slate sizes. A natural heuristic for the conditional choice model would be, once the k-slate is selected, to *order* the items on slate based their top-k or greedy scores to increase the odds that the random slate actually observed by the user contains items that induce highest expected long-term engagement.

6 User Simulation Environment

We discuss experiments with the various SLATEQ algorithms in Section 7, using a simulation environment that, though simplified and stylized, captures several essential elements of a typical recommender system that drive a need for the long/short-term tradeoffs captured by RL methods. In this section, we describe the simulation environment and models used to test SLATEQ in detail. We describe the environment setup in a fairly general way, as well as the specific instantiations used in our experiments, since the simulation environment may be of broader interest.

6.1 Document and Topic Model

We assume a set of *documents* D representing the content available for recommendation. We also assume a set of *topics* (or user interests) T that capture fundamental characteristics of interest to users; we assume topics are indexed $1, 2, \ldots |T|$. Each document $d \in D$ has an associated *topic vector* $\mathbf{d} \in [0, 1]^{|T|}$, where d_j is the degree to which d reflects topic j.

In our experiments, for simplicity, each document d has only a single topic $T(d)$, so $\mathbf{d} = \mathbf{e}_i$ for some $i \leq |T|$ (i.e., we have a one-hot encoding of the document topic). Documents are drawn from content distribution P_D over topic vectors, which in our one-hot topic experiments is simply a distribution over individual topics.

Each document $d \in D$ also has a length $\ell(d)$ (e.g., length of a video, music track or news article). This is sometime used as one factor in assessing potential user engagement. While the model supports documents of different lengths, in our experiments, we assume each document d has the same constant length ℓ.

Documents also have an *inherent quality* L_d, representing the topic-independent attractiveness to the average user. Quality varies randomly across documents, with document d's quality distributed according to $\mathcal{N}(\mu_{T(d)}, \sigma^2)$, where μ_t is a *topic-specific* mean quality for any $t \in T$. For simplicity, we assume a fixed variance across all topics. In general, quality can be estimated over time from user responses as we discuss below; but in our experiments, we assume L_d is observable to the recommender system (but not to the user *a priori*, see below). Quality may also be user-dependent, though we do not consider that here, since the focus of our stylized experiments is on the ability of our RL methods to learn average quality at the topic level. Both the topic and quality of a consumed document impact long-term user behavior (see Section 6.4 below).

In our experiments, we use $T = 20$ topics, while the precise number of documents $|D|$ is immaterial as we will see. Of these, 14 topics are low quality, with their mean quality

evenly distributed across the interval $\mu_t \in [-3, 0]$. The remaining 6 topics are high quality, with their mean quality evenly distributed across the interval $\mu_t \in [0, 3]$.

6.2 User Interest and Satisfaction Models

Users $u \in U$ have various degrees of interests in topics, ranging from -1 (completely uninterested) to 1 (fully interested), with each user u associated with an *interest vector* $\mathbf{u} \in [-1, 1]^{|T|}$. User u's interest in document d is given by the dot product $I(u, d) = \mathbf{ud}$. We assume some prior distribution P_U over user interest vectors, but user u's interest vector is dynamic, i.e., influenced by their document consumption (see below). To focus on how our RL methods learn to influence user interests and the quality of documents consumed, we treat a user's interest vector $\mathbf{u}$ as *fully observable* to the recommender system. In general, user interests are latent, and a partially observable/belief state model is more appropriate.

A user's *satisfaction* $S(u, d)$ with a consumed document d is a function $f(I(u, d), L_d)$ of user u's interest and document d's quality. While the form of f may be quite complex in general, we assume a simple convex combination $S(u, d) = (1 - \alpha)I(u, d) + \alpha L_d$. Satisfaction influences user dynamics as we discuss below.

In our experiments, a new user u's prior interest $\mathbf{u}$ is sampled uniformly from $\mathbf{u} \in [-1, 1]^{|T|}$; specifically, there is no prior correlation across topics. We use an extreme value of $\alpha = 1.0$ so that a user's satisfaction with a consumed document is fully dictated by document quality. This leaves user interest only to drive the selection of the document from the slate which we describe next.

6.3 User Choice Model

When presented with a slate of k documents, a *user choice* model impacts which document (if any) from the slate is consumed by the user. We assume that a user *can observe* any recommended document's topic prior to selection, but *cannot observe* its quality before consumption. However, the user will observe the true document quality *after* consuming it. While somewhat stylized, this treatment of topic and quality observability (from the user's) perspective is reasonably well-aligned with the situation in many recommendation domains.

The general simulation environment allows arbitrary choice functions to be defined as a function of user's state (interest vector, satisfaction) and the features of the document (topic vector, quality) in the slate. In our experiments, we use the general conditional choice model (Eq. (2)) as the main model for our RL methods. User u's interest in document d, $I(u, d) = \mathbf{ud}$, defines the document's relative appeal to the user and serves as the basis of the choice function. For slates of size k, the null document $\perp$ is always added as a $(k + 1)$st element, which (for simplicity) has a fixed utility across all users.

We also use a second choice model in our experiments, an *exponential cascade model* [Joachims, 2002], that accounts for document position on a slate. This choice model assumes "attention" is given to one document at a time, with exponentially decreasing attention given to documents as a user moves down the slate. The probability that the document in position j is inspected is $\beta_0 \beta^j$, where β_0 is a *base inspection probability* and β is the *inspection decay*.

If a document is given attention, then it is selected with a *base choice probability* $P(u, d)$; if the document in position j is not examined or selected/consumed, then the user proceeds to the $(j+1)$st document. The probability that the document in position j is consumed is:

$$P(j, A) = \beta_0 \beta^j P(u, d).$$

While we don't optimize for this model, we do run experiments in which the recommender learns a policy that assumed the general conditional choice model, but users behave according to the cascade model. In this case, the base choice probability $P(u, d)$ for a document in the cascade model is set to be its normalized probability in the conditional choice model. While the cascade model allows for the possibility of no click, even without the fictitious null document $\perp$, we keep the null document to allow the probabilities to remain calibrated relative to the conditional model. In our experiments, we use $\beta_0 = 1.0$ and $\beta = 0.65$.

6.4 User Dynamics

To allow for a non-myopic recommendation algorithm—in our case, RL methods—to impact overall user engagement positively, we adopt a simple, but natural model of *session termination*. We assume each user u has an initial *budget* B_u of time to engage with content during an extended session. This budget is not observable to the recommender system, and is randomly realized at session initiation using some prior $P(B)$.[13] Each document d consumed reduces user u's budget by the fixed document length ℓ. But after consumption, the quality of the document (partially) replenishes the used budget where the budget decreases by the fixed document length ℓ less a *bonus* $b < \ell$ that increases with the document's appeal $S(u, d)$. In effect, more satisfying documents decrease the time remaining in a session at a lower rate. In particular, for any fixed topic, documents with higher quality have a higher positive impact on cumulative engagement (reduce budget less quickly) than lower quality documents. A session ends once B_u reaches 0. Since sessions terminate with probability 1, discounting is unnecessary.

In our experiments, each user's initial budget is $B_u = 200$ units of time; each consumed document uses $\ell = 4$ units; and if a slate is recommended, but no document is clicked, 0.5 units are consumed. We set bonus $b = \frac{0.9}{3.4} \cdot \ell \cdot S(u, d)$.

The second aspect of user dynamics allows user interests to evolve as a function of the documents consumed. When user u consumes document d, her interest in topic $T(d)$ is nudged stochastically, biased slightly towards increasing her interest, but allows some chance of decreasing her interest. Thus, a recommender faces a short-term/long-term tradeoff between nudging a user's interests toward topics that tend to have higher quality at the expense of short-term consumption of user budget.

We use the following stylized model to set the magnitude of the adjustment—how much the interest in topic $T(d)$ changes—and its polarity—whether the user's interest in topic $T(d)$ increases or decreases. Let $t = T(d)$ be the topic of the consumed document d and I_t

[13] Naturally, other models that do not use terminating sessions are possible, and could emphasize amount of engagement per period.

be user u's interest in topic t prior to consumption of document d. The (absolute) change in user u's interest is $\Delta_t(I_t) = (-y|I_t| + y) \cdot -I_t$, where $y \in [0, 1]$ denotes the fraction of the distance between the current interest level and the maximum level $(1, -1)$ that the update move user u's interest. This ensures that more entrenched interests change less than neutral interests.

In our experiments we set $y = 0.3$. A positive change in interest, $I_t \leftarrow I_t + \Delta_t(I_t)$, occurs with probability $[I(u, d) + 1]/2$, and a negative change, $I_t \leftarrow I_t - \Delta_t(I_t)$, with probability $[1 - I(u, d)]/2$. Thus positive (resp., negative) interests are more likely to be reinforced, i.e., become more positive (resp., negative), with the odds of such reinforcement increasing with the degree of entrenchment.

6.5 Recommender System Dynamics

At each stage of interaction with a user, m *candidate* documents are drawn from P_D, from which a slate of size k must be selected for recommendation. This reflects the common situation in many large-scale commercial recommenders in which a variety of mechanisms are used to sub-select a small set of candidates from a massive corpus, which are in turn scored using more refined (and computationally expensive) predictive models of user engagement.

In our simulation experiments, we use $m = 10$ and $k = 3$. This small set of candidate documents and the small slate size is used to allow explicitly enumeration of all slates, which allows us to compare SLATEQ to RL methods like Q-learning that do not decompose the Q-function. In our live experiments with the YouTube platform (see Section 9), slates are of variable size and the number of candidates is on the order of $O(1000)$.

7 Empirical Evaluation: Simulations

We now describe several sets of results designed to assess the impact of the SLATEQ decomposition. Our simulation environment is implemented in a general fashion, supporting many of the general models and behaviors described in the previous sections. Our RL algorithms, both those using SLATEQ and FSQ, are implemented using Dopamine [Castro et al., 2018]. We use a standard two-tower architecture with stacked fully connected layers to represent user state and document. Updates to the Q-models are done online by batching experiences from user simulations. Each training-serving strategy is evaluated over 5000 simulated users for statistical significance. All results are within a 95% confidence interval.

7.1 Myopic vs. Non-myopic Recommendations

We first test the quality of *(non-myopic) LTV policies* learned using SLATEQ to optimize engagement ($\gamma = 1$), using a selection of the SLATEQ algorithms (SARSA vs. Q-learning, different slate optimizations for training/serving). We compare these to *myopic scoring (MYOP)* ($\gamma = 0$), which optimizes only for immediate reward, as well as a Random policy. The goal of these comparisons is to identify whether optimizing for long-term engagement

using RL (either Q-learning or 1-step policy improvement via SARSA) provides benefit over myopic recommendations.

The following table compares several key metrics of the final trained algorithms (all methods use 300K training steps):

Strategy	Avg. Return (%)	Avg. Quality (%)
Random	159.2	-0.5929
MYOP-TS	166.3 (4.46%)	-0.5428 (8.45%)
MYOP-GS	166.3 (4.46%)	-0.5475 (7.66%)
SARSA-TS	168.4 (5.78%)	-0.4908 (17.22%)
SARSA-GS	172.1 (8.10%)	-0.3876 (34.63%)
QL-TT-TS	168.4 (5.78%)	-0.4931 (16.83%)
QL-GT-GS	172.9 (8.61%)	-0.3772 (36.38%)
QL-OT-TS	169.0 (6.16%)	-0.4905 (17.27%)
QL-OT-GS	173.8 (9.17%)	-0.3408 (42.52%)
QL-OT-OS	174.6 (9.67%)	-0.3056 (48.46%)

The LTV methods (SARSA and Q-learning) using SLATEQ offer overall improvements in average return per user session. The magnitude of these improvements only tells part of the story: we also show percentage improvements relative to Random are shown in parentheses—Random gives a sense of the baseline level of cumulative reward that can be achieved without any user modeling at all. For instance, relative to the random baseline, QL-OT-GS provides a provides a 105.6% greater improvement than MYOP. The LTV methods all learn to recommend documents of much higher quality than MYOP, which has a positive impact on overall session length, which explains the improved return per user.

We also see that LP-based slate optimization during training (OT) provides improvements over top-k and greedy optimization (TT, GT) in Q-learning when comparing similar serving regimes (e.g., QL-OT-GS vs. QL-GT-GS , and QL-OT-TS vs. QL-TT-TS). Optimal serving (OS) also shows consistent improvement over top-k and greedy serving—and greedy serving (GS) improves significantly over top-k serving (TS)—when compared under the same training regime. However, the combination of optimal training and top-k or greedy serving performs well, and is especially useful when serving latency constraints are tight, since optimal training is generally done offline.

Finally, optimizing using Q-learning gives better results than on-policy SARSA (i.e., one-step improvement) under comparable training and serving regimes. But SARSA itself has significantly higher returns than MYOP, demonstrating the value of on-policy RL for recommender systems. Indeed, repeatedly serving-then-training (with some exploration) using SARSA would implement a natural, continual policy improvement. These results demonstrate, in this simple synthetic recommender system environment, that using RL to plan long-term interactions can provide significant value in terms of overall engagement.

7.2 SLATEQ vs. Holistic Optimization

Next we compare the quality of policies learned using the SLATEQ decomposition to FSQ, the non-decomposed Q-learning method that treats each slate atomically as a single action. We set $|T| = 20$, $m = 10$, and $k = 3$ so that we can enumerate all $\binom{10}{3}$ slates for FSQ maximization. Note that the Q-function for FSQ requires representation of all $\binom{20}{3} = 1140$ slates as actions, which can impede both exploration and generalization. For SLATEQ we test only SARSA-TS (since this is the method tested in our live experiment below). The following table shows our results:

	Avg. Return (%)	Avg. Quality (%)
Random	160.6	-0.6097
FSQ	164.2 (2.24%)	-0.5072 (16.81%)
SARSA-TS	170.7 (6.29%)	-0.5340 (12.41%)

While FSQ, which is an off-policy Q-learning method, is guaranteed to converge to the optimal slate policy in theory with sufficient exploration, we see that, even using an *on-policy method* like SARSA with a single step of policy improvement, SLATEQ methods perform significantly better than FSQ, offering a 180% greater improvement over Random than FSQ. This is the case despite SLATEQ using no additional training-serving iterations to continue policy improvement. This is due to the fact that FSQ must learn Q-values for 1140 distinct slates, making it difficult to explore and generalize. FSQ also takes roughly 6X the training time of SLATEQ over the same number of events. These results demonstrate the considerable value of the SLATEQ decomposition.

Improved representations could help FSQ generalize somewhat better, but the approach is inherently unscalable, while SLATEQ suffers from no such limitations (see live experiment below). Interestingly, FSQ does converge quickly to a policy that offers recommendations of greater average quality than SLATEQ, but fails to make an appropriate tradeoff with user interest.

7.3 Robustness to User Choice

Finally, we test the robustness of SLATEQ to changes in the underlying user choice model. Instead of the assumed choice model defined above, users select items from the recommended slate using a simple (exponential) *cascade model*, where items on the slate are inspected from top-to-bottom with a position-specific probability, and consumed with probability proportional to $I(u, d)$ if inspected. If not consumed, the next item is inspected. Though users act in this fashion, SLATEQ is trained using the original conditional choice model and the same decomposition is also used to optimize slates at serving time.

The following table shows results:

Strategy	Avg. Return (%)	Avg. Quality (%)
Random	159.9	-0.5976
MYOP-TS	163.6 (2.31%)	-0.5100 (14.66%)
SARSA-TS	166.8 (4.32%)	-0.4171 (30.20%)
QL-TT-TS	166.5 (4.13%)	-0.4227 (29.27%)
QL-OT-TS	167.5 (4.75%)	-0.3985 (33.32%)
QL-OT-OS	167.6 (4.82%)	-0.3903 (34.69%)

SLATEQ continues to outperform MYOP, even when the choice model does not accurately reflect the true environment, demonstrating its relative robustness. SLATEQ can be used with other choice models. For example, SLATEQ can be trained by assuming the cascade model, with only the optimization formulation requiring adaptation (see our discussion in Section 5.4). But since any choice model will generally be an *approximation* of true user behavior, this form of robustness is critical.

Notice that QL-TT and SARSA have inverted relative performance compared to the experiments above. This is due to the fact that Q-learning exploits the (incorrect) choice model to optimize during training, while SARSA, being on-policy, only uses the choice model to compute expectations at serving time. This suggests that an on-policy control method like SARSA (with continual policy improvement) may be more robust than Q-learning in some settings.

8 A Practical Methodology

The deployment of a recommender system using RL or TD methods to optimize for long-term user engagement presents a number of challenges in practice. In this section, we identify several of these and suggest practical techniques to resolve them, including ways in which to exploit an existing *myopic*, item-level recommender to facilitate the deployment of a non-myopic system.

Many (myopic) item-level recommender systems [Liu et al., 2009, Covington et al., 2016] have the following components:

(i) *Logging* of impressions and user feedback;

(ii) *Training* of some regression model (e.g., DNN) to predict user responses for user-item pairs, which are then aggregated by some scoring function;

(iii) *Serving* of recommendations, ranking items by score (e.g., returning the top k items for recommendation).

Such a system can be exploited to quickly develop a non-myopic recommender system based on Q-values, representing some measure of long-term engagement, by addressing several key challenges.

8.1 State Space Construction

A critical part of any RL modeling is the design of the state space, that is, the development of a set of features that adequately capture a user's past history to allow prediction of long-term value (e.g., engagement) in response to a recommendation. For the underlying process to be a MDP, the feature set should be (at least approximately) predictive of immediate user response (e.g., immediate engagement, hence *reward*) and self-predictive (i.e., summarizes user history in a way that renders the implied dynamics Markovian).

The features of an extant myopic recommender system typically satisfy both of these requirements, meaning that an RL or TD model can be built using the same logged data (organized into trajectories) and the same featurization. The engineering, experimentation and experience that goes into developing state-of-the-art recommender systems means that they generally capture (almost) all aspects of history required to predict immediate user responses (e.g., pCTR, listening time, other engagement metrics); i.e., they form a sufficient statistic. In addition, the core input features (e.g., static user properties, summary statistics of past behavior and responses) are often self-predictive (i.e., no further history could significantly improve next state prediction). This fact can often be verified by inspection and semantic interpretation of the (input) features. Thus, using the existing state definition provides a natural, practical way to construct TD or RL models. We provide experimental evidence below to support this assertion in Section 9.

8.2 Generalization across Users

In the MDP model of a recommender system, each user should be viewed as a separate environment or separate MDP. However, it is critical to allow for generalization across users, since few if any users generate enough experience to allow reasonable recommendations otherwise. Of course, such generalization is a hallmark of almost any recommender system. In our case, we must generalize the (implicit) MDP dynamics across users. The state representation afforded by an extant myopic recommender system is already intended to do just this, so by learning a Q-function that depends on the same user features and the myopic system, we obtain the same form of generalization.

8.3 User Response Modeling

As noted in Sections 4 and 5, SLATEQ takes advantage of some measure of immediate item appeal or utility (conditioned on a specific user or state) to determine user choice behavior. In practice, since myopic recommender systems often predict these immediate responses, for example, using pCTR models, we can use these models directly to assess the immediate appeal $v(s, i)$ required by our SLATEQ choice model. For instance, we can use a myopic model's pCTR predictions directly as a (unnormalized) choice probabilities for items in a slate, or we can use the logits of such a model in the conditional logit choice model. Furthermore, by using the same state features (see above), it is straightforward to build a multi-task model [Zhang and Yang, 2017] that incorporates our long-term engagement

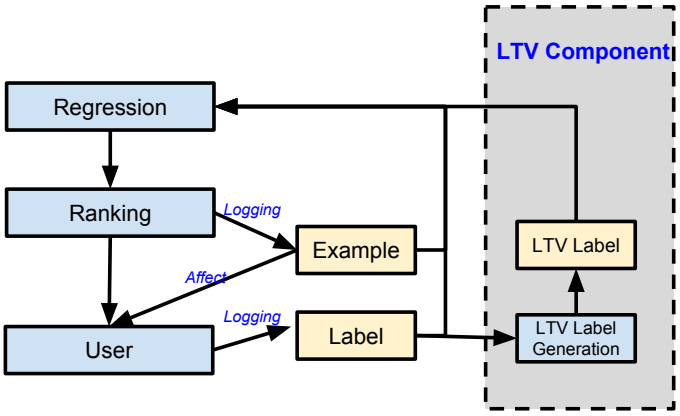

Figure 1: Schematic View of a Non-myopic Recommender Training System

prediction with other user response predictions.

8.4 Logging, Training and Serving Infrastructure

The training of long-term values $Q^\pi(s, a)$ requires logging of user data, and live serving of recommendations based on these LTV scores. The model architecture we detail below exploits the same logging, (supervised) training and serving infrastructure as used by the myopic recommender system.

Fig. 1 illustrates the structure of our LTV-based recommender system—here we focus on SARSA rather than Q-learning, since our long-term experiment in Section 9 uses SARSA. In myopic recommender systems, the regression model predicts immediate user response (e.g., clicks, engagement), while in our non-myopic recommender system, label generation provides LTV labels, allowing the regressor to model $\overline{Q}^\pi(s, a)$.

Models are trained periodically and pushed to the server. The ranker uses the latest model to recommend items and logs user feedback, which is used to train new models. Using LTV labels, iterative model training and pushing can be viewed as a form of *generalized policy iteration* [Sutton and Barto, 1998]. Each trained DNN represents the value of the policy that generated the prior batch of training data, thus training is effectively *policy evaluation*. The ranker acts greedily with respect to this value function, thus performing *policy improvement*.

LTV label generation is similar to DQN training [Mnih et al., 2015]. A main network learns the LTV of individual items, $\overline{Q}^\pi(s, a)$—this network is easily extended from the existing myopic DNN. For stability, bootstrapped LTV labels (Q-values) are generated using a separate *label network*. We periodically copy the weights of the main network to the label network and use the (fixed) label network $\overline{Q}_{label}(s, a)$ to compute LTV labels between copies. LTV labels are generated using Eq. (19).

9 Empirical Evaluation: Live Experiments

We tested the SLATEQ decomposition—specifically, the SARSA-TS algorithm, on YouTube (`https://www.youtube.com/`), a large-scale video recommender with $O(10^9)$ users and $O(10^8)$ items in its corpus. The system is typical of many practical recommender systems with two main components. A *candidate generator* retrieves a small subset (hundreds) of items from a large corpus that best match a user context. The *ranker* scores/ranks candidates using a DNN with both user context and item features as input. It optimizes a combination of several objectives (e.g., clicks, expected engagement, several other factors).

The extant recommender system's policy is *myopic*, scoring items for the slate using their *immediate* (predicted) expected engagement. In our experiments, we replace the myopic engagement measure with an *LTV estimate* in the ranker scoring function. We retain other predictions and incorporate them into candidate scoring as in the myopic model. Our non-myopic recommender system maximizes *cumulative* expected engagement, with user trajectories capped at N days. Since homepage visits can be spaced arbitrarily in time, we use time-based rather than event-based discounting to handle credit assignment across large time gaps. If consecutive visits occur at times t_1 and t_2, respectively, the relative discount of the reward at t_2 is $\gamma^{(t_2-t_1)/c}$, where c is a parameter that controls the time scale for discounting.

Our model extends the myopic ranker using the practical methodology outlined in Section 8. Specifically, we learn a multi-task feedforward deep network [Zhang and Yang, 2017], which learns $\overline{Q}(s,i)$, the predicted long-term engagement of item i (conditional on being clicked) in state s, as well as the immediate appeal $v(s,i)$ for pCTR/user choice computation (several other response predictions are learned, which are identical to those used by the myopic model). The multi-task feedforward DNN network has 4 hidden layers of sizes 2048, 1024, 512, 256; and used ReLU activation functions on each of the hidden layers. Apart from the LTV/Q-value head, other heads include pCTR, and other user responses. To validate our methodology, the DNN structure and all input features are identical to the production model which optimizes for short-term (myopic) immediate reward. The state is defined by user features (e.g., user's past history, behavior and responses, plus static user attributes). This also makes the comparison with the baseline fair.

The full training algorithm used in our live experiment is shown in Algorithm 1. The model is trained using TensorFlow in a distributed training setup [Abadi et al., 2015] using stochastic gradient descent. We train on-policy over pairs of consecutive start page visits, with LTV labels computed using Eq. 19, and use top-k optimization for serving—i.e., we test SARSA-TS. The existing myopic recommender system (baseline) also builds slates greedily—i.e., MYOP-TS.

We note that at serving time, we don't just choose the slate using the top-k method, we also *order* the slate presented to the user according to the item scores $v(s,i)\overline{Q}^{\pi}(s,i)$ for each item i (at state s). The reason for this is twofold. First, we expect that the user experience is positively impacted by placing more appealing items, that are likely to induce longer-term engagement, earlier in the slate. Second, the scrolling nature of the interface means that the

Algorithm 1 On-policy SLATEQ for Live Experiments

Parameters:
- T: the number of iterations.
- M: the interval to update label network.
- γ: discount rate.
- θ_{main}: the parameter for the main neural network.
- $\overline{Q}_{main}$: that predicts items' long-term value.
- θ_{label}: the parameter for the label neural network $\overline{Q}_{label}$.
- θ_{pctr}: the parameter for the neural network that predicts items' pCTR.

Input: $D_{training} = (s, A, C, L_{myopic}, s', A')$: the training data set.
- s: current state features
- $A = (a_1, ..., a_k)$: recommended slate of items in current state; a_i denotes item features
- $C = (c_1, ..., c_k)$: c_i denotes whether item a_i is clicked
- $L_{myopic} = (l^1_{myopic}, ..., l^k_{myopic})$: myopic (immediate) labels
- s': next state features
- $A' = (a'_1, ..., a'_k)$: recommended slate of items in next state.

Output: Trained Q-network $\overline{Q}_{main}$ that predicts items' long-term value.
Initialization $\theta_{label} = 0$, θ_{main} randomly, θ_{pctr} randomly
for $i = 1 \dots T$ **do**
  if $i \bmod M = 0$ **then**
    $\theta_{label} \leftarrow \theta_{main}$
  end if
  for each example $(s, A, C, L_{myopic}, s', A') \in D_{training}$ **do**
    for each item $a_i \in A$ **do**
      update θ_{pctr} using click label c_i
      if a_i is clicked **then**
        probability: $pCTR(s', a'_i, A') \leftarrow pCTR(s', a'_i) / \sum_{a'_i \in A} pctr(s', a'_i)$
        LTV label: $l^i_{ltv} \leftarrow l^i_{myopic} + \sum_{a'_i \in A'} pCTR(s', a'_i, A') \overline{Q}_{label}(s', a'_i)$
        update θ_{main} using LTV label l^i_{ltv}
      end if
    end for
  end for
end for

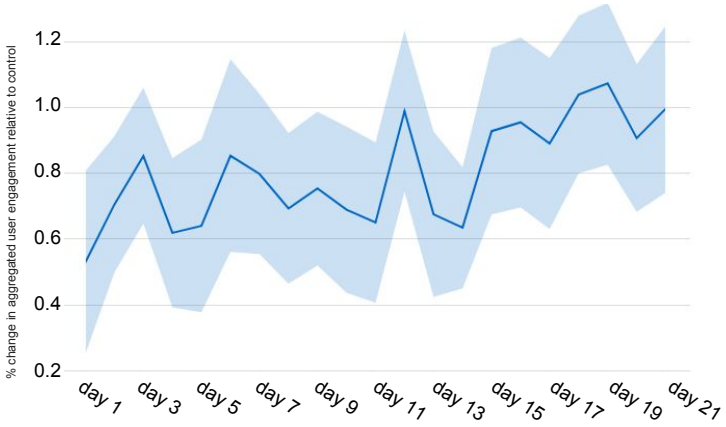

Figure 2: Increase in user engagement over the baseline. Data points are statistically significant and within 95% confidence intervals.

slate size k is not fixed at serving time—the number of inspected items varies per user-event (see discussion in Section 5.4).

We experimented with live traffic for three weeks, treating a small, but statistically significant, fraction of users to recommendations generated by our SARSA-TS LTV model. The control is a highly-optimized production machine learning model that optimizes for immediate engagement (MYOP-TS). Fig. 2 shows the percentage increase in aggregate user engagement using LTV over the course of the experiment relative to the control, and indicates that our model outperformed the baseline on the key metric under consideration, consistently and significantly. Specifically, users presented recommendations by our model had sessions with greater engagement time relative to baseline.

Fig. 3 shows the change in distribution of cumulative engagement originating from items at different positions in the slate. Recall that the number of items viewed in any user-event varies, i.e., experienced slates are of variable size and we show the first ten positions in the figure. The results show that the users under treatment have more engaging sessions (larger LTVs) from items ranked higher in the slate compared to users in the control group, which suggests that top-k slate optimization performs reasonably in this domain.[14]

10 Conclusion

In this work, we addressed the problem of optimizing long-term user engagement in slate-based recommender systems using reinforcement learning. Two key impediments to the use of RL in large-scale, practical recommenders are (a) handling the combinatorics of slate-based action spaces; and (b) constructing the underlying representations.

[14]The apparent increase in expected engagement at position 10 is a statistical artifact due to the small number of events at that position: the number of observed events at each position decreases roughly exponentially, and position 10 has roughly two orders of magnitude fewer observed events than any of the first three positions.

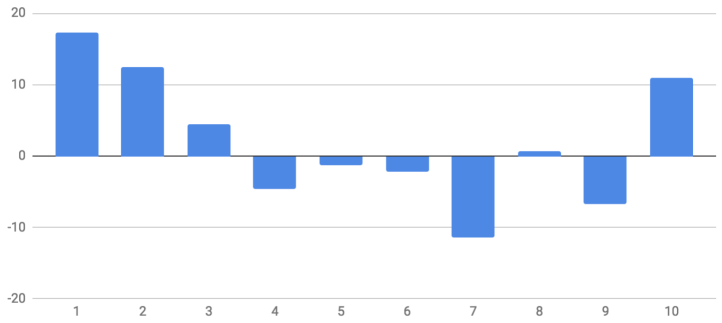

Figure 3: Percentage change in long-term user engagement vs. control (y-axis) across positions in the slate (x-axis). Top 3 positions account for approximately 95% of engagement.

To handle the first, we developed SLATEQ, a novel decomposition technique for slate-based RL that allows for effective TD and Q-learning using LTV estimates for individual items. It requires relatively innocuous assumptions about user choice behavior and system dynamics, appropriate for many recommender settings. The decomposition allows for effective TD and Q-learning by reducing the complexity of generalization and exploration to that of learning for individual items—a problem routinely addressed by practical myopic recommenders. Moreover, for certain important classes of choice models, including the conditional logit, the slate optimization problem can be solved tractably using optimal LP-based and heuristic greedy and top-k methods. Our results show that SLATEQ is relatively robust in simulation, and can scale to large-scale commercial recommender systems like YouTube.

Our second contribution was a practical methodology for the introduction of RL to extant, myopic recommenders. We proposed the use of existing myopic models to bootstrap the development of Q-function-based RL methods, in a way that allows the substantial reuse of current training and serving infrastructure. Our live experiment in YouTube recommendation exemplified the utility of this methodology and the scalability of SLATEQ. It also demonstrated that using LTV estimation can improve user engagement significantly in practice.

There are a variety of future research directions that can extend the work here. First, our methodology can be extended by relaxing some of the assumptions we made regarding the interaction between user choice and system dynamics. For instance, we are interested in models that allow unconsumed items on the slate to influence user latent state and choice models that allow for multiple items on a slate to be used/clicked. Further analysis of, and the development of corresponding optimization procedures for, additional choice models using SLATEQ remains of intense interest (e.g., hierarchical model such as nest logit). In a related vein, methods for simultaneous learning of choice models, or their parameters, while learning Q-values would be of great practical value. Finally, the simulation environment has the potential to serve as a platform for additional research on the application of RL to recommender systems. We hope to release a version of it to the research community in the near future.

Acknowledgments. Thanks to Larry Lansing for system optimization and the IJCAI-2019 reviewers for helpful feedback.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. Learning a deep listwise context model for ranking refinement. In *Proceedings of the 41st Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR-18)*, pages 135–144, 2018.

Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. Seq2slate: Re-ranking and slate optimization with rnns. `arXiv:1810.02019 [cs.IR]`, 2018.

Dimitri P. Bertsekas and John. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena, Belmont, MA, 1996.

Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. Active collaborative filtering. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 98–106, Acapulco, 2003.

Craig Boutilier, Alon Cohen, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. Planning and learning with stochastic action sets. In *Proceedings of the Twenty-seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 4674–4682, Stockholm, 2018.

Jack S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, Madison, WI, 1998.

Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-14)*, pages 1433–1452, 2014.

Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1–2):67–119, 2014.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A research framework for deep reinforcement learning. `arXiv:1812.06110 [cs.LG]`, 2018.

Abraham Charnes and William W. Cooper. Programming with linear fractional functionals. *Naval Research Logistics Quarterly*, 9(3-4):181–186, 1962.

Kyle D. Chen and Warren H. Hausman. Mathematical properties of the optimal product line selection problem using choice-based conjoint analysis. *Management Science*, 46(2): 327–332, 2000.

Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi. Top-k off-policy correction for a REINFORCE recommender system. In *12th ACM International Conference on Web Search and Data Mining (WSDM-19)*, pages 456–464, Melbourne, Australia, 2018.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10, Boston, 2016.

Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. Reinforcement learning-based recommender system using biclustering technique. `arXiv:1801.05532 [cs.IR]`, 2018.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198, Boston, 2016.

Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM-08)*, pages 87–94. ACM, 2008.

Peter Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

Uriel Feige. A threshold of ln(n) for approximating set cover. *Journal of the ACM*, 45(4): 634–652, 1998.

Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, and Xiaohui Ye. Horizon: Facebook's open source applied reinforcement learning platform. `arXiv:1811.00260 [cs.LG]`, 2018.

Carlos A. Gomez-Uribe and Neil Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4):13:1–13:19, 2016.

Assaf Hallak, Yishay Mansour, and Elad Yom-Tov. Automatic representation for lifetime value recommender systems. `arXiv:1702.07125 [stat.ML]`, 2017.

Ruining He and Julian McAuley. Fusing similarity models with Markov chains for sparse sequential recommendation. In *Proceedings of the IEEE International Conference on Data Mining (ICDM-16)*, Barcelona, 2016.

Dorothee Honhon, Sreelata Jonnalagedda, and Xiajun Amy Pan. Optimal algorithms for assortment selection under ranking-based consumer choice models. *Manufacturing and Service Operations Management*, 14(2):279–289, 2012. doi: 10.1287/msom.1110.0365. URL http://pubsonline.informs.org/doi/abs/10.1287/msom.1110.0365.

Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 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, 2019. To appear.

Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. Music personalization at Spotify. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys16)*, pages 373–373, Boston, Massachusetts, USA, 2016.

Ray Jiang, Sven Gowal, Timothy A. Mann, and Danilo J. Rezende. Beyond greedy ranking: Slate optimization via List-CVAE. In *Proceedings of the Seventh International Conference on Learning Representations (ICLR-19)*, New Orleans, 2019.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 133–142, 2002.

Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent Dirichlet allocation for tag recommendation. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys09)*, pages 61–68, New York, 2009.

Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *Proceedings of the Thirty-second International Conference on Machine Learning (ICML-15)*, pages 767–776, 2015.

Dung D. Le and Hady W. Lauw. Indexable Bayesian personalized ranking for efficient top-k recommendation. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM-17)*, pages 1389–1398, 2017.

Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

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

R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. Wiley, 1959.

Victor Martínez-de Albéniz and Guillaume Roels. Competing for shelf space. *Production and Operations Management*, 20(1):32–46, 2011. doi: 10.1111/j.1937-5956.2010.01126.x. URL http://dx.doi.org/10.1111/j.1937-5956.2010.01126.x.

Daniel McFadden. Conditional logit analysis of qualitative choice behavior. In Paul Zarembka, editor, *Frontiers in Econometrics*, pages 105–142. Academic Press, 1974.

Rishabh Mehrotra, Mounia Lalmas, Doug Kenney, Thomas Lim-Meng, and Golli Hashemian. Jointly leveraging intent and interaction signals to predict user satisfaction with slate recommendations. In *2019 World Wide Web Conference (WWW'19)*, pages 1256–1267, San Francisco, 2019.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep RL. `arXiv:1705.05035 [cs.LG]`, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Yashar Moshfeghi, Benjamin Piwowarski, and Joemon M. Jose. Handling data sparsity in collaborative filtering using emotion and semantic based features. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-11)*, pages 625–634, Beijing, 2011.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized Markov chains for next-basket recommendation. In *Proceedings of the 19th International World Wide Web Conference (WWW-10)*, pages 811–820, Raleigh, NC, 2010.

Gavin A. Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical Report Technical Report TR166, University of Cambridge, Department of Engineering, Cambridge, UK, 1994.

Paat Rusmevichientong and Huseyin Topaloglu. Robust assortment optimization in revenue management under the multinomial logit choice model. *Operations Research*, 60(4): 865–882, 2012.

Nachiketa Sahoo, Param Vir Singh, and Tridas Mukhopadhyay. A hidden Markov model for collaborative filtering. *Management Information Systems Quarterly*, 36(4), 2012.

Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*, pages 1257–1264, Vancouver, 2007.

Cornelia Schön. On the optimal product line selection problem with price discrimination. *Management Science*, 56(5):896–902, 2010.

Guy Shani, David Heckerman, and Ronen I. Brafman. An MDP-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

Nathan Srebro, Jason Rennie, and Tommi Jaakkola. Maximum margin matrix factorization. In *Advances in Neural Information Processing Systems 17 (NIPS-2004)*, pages 1329–1336, Vancouver, 2004.

Peter Sunehag, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin. Deep reinforcement learning with attention for slate Markov decision processes with high-dimensional states and actions. `arXiv:1512.01124 [cs.AI]`, 2015.

Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 9 (NIPS-96)*, pages 1038–1044, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems 30 (NIPS-17)*, pages 3632–3642, Long Beach, CA, 2017.

Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: A reinforcement learning approach. In *Proceedings of the First ACM Conference on Recommender Systems (RecSys07)*, pages 113–120, Minneapolis, 2007. ACM.

Kalyan Talluri and Garrett van Ryzin. Revenue management under a general discrete choice model of consumer behavior. *Management Science*, 50(1):15–33, 2004.

Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22, Boston, 2016.

Georgios Theocharous, Philip S. Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 1806–1812, Buenos Aires, 2015.

Kenneth E. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, Cambridge, 2009.

Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, pages 2643–2651, Lake Tahoe, NV, 2013.

Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected SARSA. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, 2009.

Paolo Viappiani and Craig Boutilier. Optimal Bayesian recommendation sets and myopically optimal choice query sets. In *Advances in Neural Information Processing Systems 23 (NIPS)*, pages 2352–2360, Vancouver, 2010.

Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the Twenty-first ACM International Conference on Knowledge Discovery and Data Mining (KDD-15)*, pages 1235–1244, Sydney, 2015.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. Practical diversified recommendations on YouTube with determinantal point processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM18)*, pages 2165–2173, Torino, Italy, 2018.

Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM-17)*, pages 495–503, Cambridge, UK, 2017.

Yu Zhang and Qiang Yang. A survey on multi-task learning. `arXiv:1707.08114 [cs.LG]`, 2017.

Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys-18)*, pages 95–103, Vancouver, 2018.