Buyer-Side Recommendation Agents

Andrew Qin
Landon Butler
Yigit Efe Erginbas
Kannan Ramchandran

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-118
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-118.html

May 16, 2025

Buyer-Side Recommendation Agents

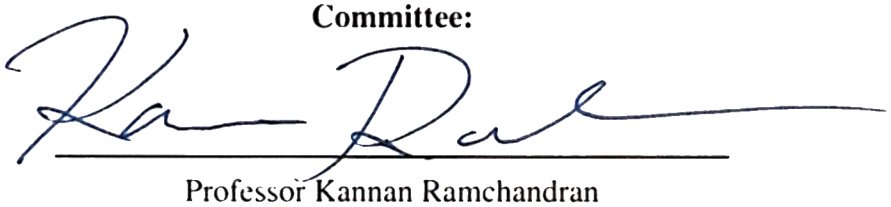
by Andrew Qin

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

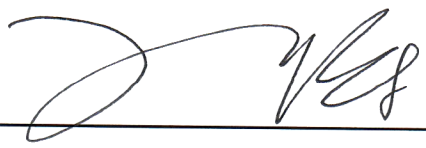


Professor Kannan Ramchandran
Research Advisor

5/16/25

(Date)

* * * * * * *



Professor Jiantao Jiao
Second Reader

05/16/2025

(Date)

Buyer-Side Recommendation Agents

by
Andrew Qin

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science
in
Electrical Engineering and Computer Sciences
in the
Graduate Division
of the
University of California, Berkeley

Committee in charge:
Professor Kannan Ramchandran, Advisor
Professor Jiantao Jiao, Second Reader

Spring 2025

Abstract

In large-scale marketplaces, recommender systems are traditionally optimized from the platform's perspective, aiming to maximize revenue or user purchase-rates. In this setting, due to the computational asymmetry, users are price-takers who must rely on recommender systems to suggest goods. We outline and study *recommendation agents*: systems designed to model and maximize an individual buyer's utility. We apply candidate retrieval, sequential recommendation, and discrete choice modeling to capture contextual user desires and price sensitivities. Our system is evaluated on a semi-synthetic dataset based on e-commerce purchase behavior. We conclude with a discussion on the construction of markets over many recommendation agents, allowing for synchronous price negotiation, effectively simulating a logit-demand price competition setting, which opens opportunities for more efficient marketplace dynamics.

Dedication

To my parents.

Contents

Acknowledgements

I am deeply grateful to my advisor, Professor Kannan Ramchandran, for his warmth, guidance, and continuous support throughout my academic journey. He has consistently provided valuable insights, from the time as a student in his class to my work in his research group. I want to express my sincere gratitude to Landon Butler for his kind mentorship, thoughtful advice, and generous time. I sincerely appreciate Yigit Efe Erginbas for his constant support and our insightful discussions. I am also grateful to Professor Jiantao Jiao for his valuable time as my second reviewer. Lastly, I want to thank my friends and family for always being there.

The work presented in this thesis was co-authored with Landon Butler with additional contributions from Yigit Efe Erginbas, under the supervision of Professor Kannan Ramchandran.

Chapter 1

Introduction

1.1 Motivation

Large-scale markets, such as e-commerce platforms, commonly apply recommender systems to suggest items to potential buyers. Typically, these recommender systems are designed to maximize platform-centric objectives, such as user retention or revenue. However, these goals do not necessarily align with the welfare of consumers, or the total welfare of the market participants.

For example, Ferreira et al. [1] conducted a randomized experiment studying recommender systems in a video-on-demand market. They found that profit-maximizing recommender systems, by exploiting consumer price-insensitivity toward recommended products, can lower consumer surplus and total market welfare from their maximum achievable values.

Such outcomes highlight a deeper issue: large-scale markets often fail to reach efficient market equilibria—idealized states where demand meets supply in a way that maximizes overall market welfare [2], [3]. These equilibria have been a central object of study in economics for their desirable efficiency properties, such as Pareto optimality, or their guaranteed existence under certain conditions [4].

However, traditional microeconomic theory often relies on assumptions that rarely hold in practice, such as perfect rationality (consumers always make utility-maximizing decisions), complete information (full knowledge of all available goods and prices), and frictionless transactions (no cost or delays in collecting information or making exchanges). In large-scale markets, individual buyers face challenges in meeting these assumptions, since they cannot feasibly explore vast item spaces to make fully rational choices.

Consequently, consumers rely on recommender systems. This creates a computational asymmetry between the platform and consumers, and potentially undermines the market's ability to converge toward efficient equilibria.

Recent research has explored algorithmic approaches as a way to relax common assumptions in microeconomics. For instance, Liu et al. [5] analyzed matching markets where participants iteratively learn their preferences, modeling the probabilistic uncertainty present in real-world settings. Their application of multi-armed bandits extends Gale and Shapley's seminal work on matching markets to settings where participants don't have perfect information about their preferences.

Buyer-side recommendation agents emerge as a natural solution for large-scale markets. Suppose each buyer is represented in the marketplace by an agent that effectively

approximates a utility function measuring the goods and prices the buyer is interested in. Furthermore, suppose each agent's sole objective is to maximize the utility of the buyer. Then, we can envision a more balanced market where both buyers and sellers interact through computational proxies. This could help markets achieve more efficient outcomes, even across large item spaces and complex preference landscapes.

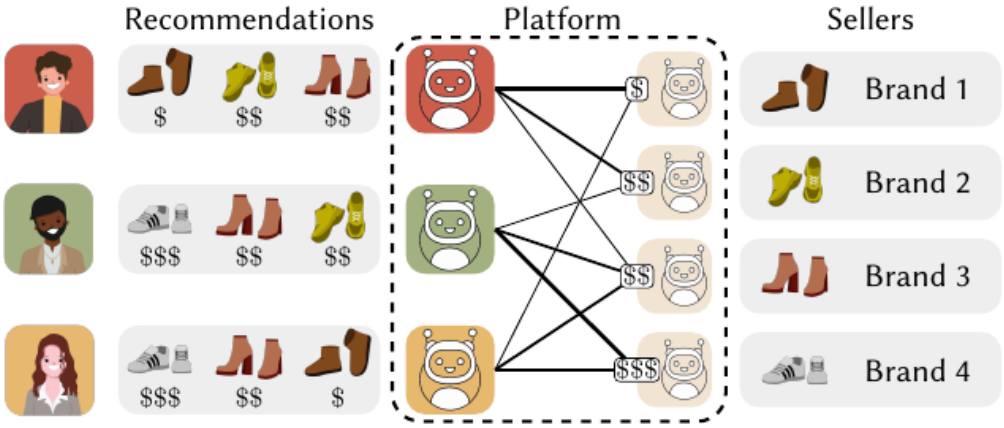


Figure 1: A depiction of a two-sided market where both buyers and sellers interact through agents who are able to negotiate prices over a large item space.

In this work, we investigate how buyer-side recommendation agent systems can be implemented using standard recommender system techniques. Our approach offers practical pathways toward efficient algorithmically-mediated marketplaces.

1.2 Recommendation Agent Proposal

Our goal is to build a recommendation agent that has learned its user's preference function and can subsequently search for goods and negotiate prices on the user's behalf. To achieve this, we propose that the recommendation agent should satisfy the following criteria:

- **Domain Agnosticism**: The agent should be able to handle cross-domain usage in order to have frequent use by a given user.
- **Price Sensitivity**: The agent's model of the user's utility should vary continuously with price—that is, distinguish between prices p and $p + \varepsilon$—so that the agent can negotiate prices over a continuous spectrum.
- **Sequential Updating**: The agent should incrementally adapt towards the user's idiosyncratic preferences over time, updating its model with each interaction.
- **Query Awareness**: The agent should consider the user's current query as context for the recommendation, because a user's contextual desire determines how price-sensitive the user is, and which items are reasonable to negotiate for.

To satisfy these requirements, we propose RecAgent, a multi-stage recommendation agent system which learns from sequences of queries and purchases. The first stage is a search model, which identifies semantically similar candidate items to the user's query. The second is

a sequential recommendation step, which narrows down the candidate items to a small set of items with greater substitutability. The third is a discrete choice model, which re-ranks the items according to their prices, adjusting price sensitivity to the query. The fourth is a feedback stage, which adjusts a learnable idiosyncratic price sensitivity parameter based on the user's final choice.

We particularly highlight the choice of incorporating price sensitivity over only a small set of items. The rationale behind this approach can be demonstrated by the following thought experiment: suppose the user wants to buy a toy boat, and a real boat is on a large discount. If the agent has learned non-trivial price sensitivity, the real boat may now have high positive utility due to the large discount in price-space. But because of the user's contextual desire, the recommendation agent shouldn't suggest buying the real boat. Therefore, identifying a set of substitute goods to compare prices over is key to our problem.

1.3 Relevant Literature

The following is a survey of relevant literature to our proposed framework.

1.3.1 Sequential Recommendation

Sequential recommendation has become a popular setting for modern recommender systems, often addressed with sequential neural networks. The objective is typically to predict a user's next purchase or click based on their prior behavior. Therefore, recommendation tasks can be framed as a "next-item" prediction given a user's purchase history or browsing history, analogous to next-token prediction in language modeling.

SASRec [6] applied the autoregressive transformer architecture [7] to sequential user histories of purchases, treating each item in the sequence as a token. Query-SeqRec [8] extended this to incorporate a user's query in searching for their next product. Specifically, they modify the transformer's input sequence to also contain natural language queries, whose embeddings are given by a pooling over a word embedding model.

Other sequential neural network architectures have been applied and studied in the sequential recommendation setting. For example, the BERT [9] and Mamba [10] sequential architectures have been applied by BERT4Rec [11] and Mamba4Rec [12], respectively. In this work, we choose to focus on autoregressive transformer architectures for simplicity.

1.3.2 Consumer Price Sensitivity

Several recent works have studied recommender systems that incorporate price.

Wang et al. [13] propose recommender systems that consider the user's utility, modeling diminishing returns and price sensitivity, applying their techniques with singular value decomposition (SVD). SHOPPER [14] is a sequential model over shopping trips which considers price through a discrete choice modeling lens, and can also identify substitute and complement goods. Price-aware Recommendation with Graph Convolutional Networks [15] incorporates price sensitivity to predict future item interactions for a given user by discretizing

and embedding price levels. Wan et al. [16] study the problem of modeling preferences based on shopping data, incorporating price sensitivity in their matrix factorization model in a linear fashion.

Our work differs in learning substitutes and price sensitivity based on a user's contextual semantic interest, studying cross-domain data, and our goal of adjusting our system to its user in an online fashion.

1.3.3 Two-sided Agent Markets

Previous work has studied settings related to our plan for two-sided agent-mediated markets. For instance, Greenwald and Kephart [17] hypothesize that "pricebots" (revenue-maximizing pricing agents) may arise in response to "shopbots" (agents that help buyers navigate large product spaces and their prices), and study the dynamics that arise out of their interactions.

Such revenue-maximizing pricing agents have also been studied recently, such as analysis given by Erginbas et al. [18]. There, sellers are modeled revenue-maximizers who are given the task of serving a set of recommended items and prices to users given contextual desires. Buyers, on the other hand, are modeled as following a multinomial logistic model (MNL) [19]. Then, the sellers must learn optimal (in the sense of regret) recommendations and pricing from the feedback given by purchases, or lack thereof, over the recommended set. This problem setting is similar to ours—we also want to learn optimal recommendations given purchase feedback over small recommendation sets, assuming an MNL model. However, we focus on the buyer side, and work towards creating a practical and empirically verifiable implementation.

1.4 Outline

In this thesis, we first construct a dataset that can simulate an environment with the information needed for RecAgent. Then, we outline our model and experiments. We conclude with a discussion on two-sided agent-mediated markets.

Chapter 2

Dataset Construction

Since our setting requires queries, varying prices, and cross-domain purchases, we curated a custom dataset. Specifically, we constructed a semi-synthetic Amazon purchase dataset enriched with queries and prices.

2.1 Data Curation

We began with Amazon review data from Ni et al. [20], which contains reviews for Amazon products. It has metadata for each review and purchased item, such as the time of each review, and item descriptions. We use reviews from the *Instruments*, *Movies and TV*, and *Video Games* categories. These categories were selected to represent product types with different purchase frequencies and price ranges.

For our sequential modeling component, we needed sequential purchase sequences. Therefore, we constructed cross-domain purchase histories by temporally ordering a user's reviews, using the time of review data as proxies for purchase times. Following prior work [6], we applied a 5-core filtering criterion: each user in our dataset must have at least 5 purchases, and each item must have been purchased at least 5 times. This mitigates scenarios where we are evaluating our model's ability to predict a purchase for an item it hasn't seen during training, for instance.

2.1.1 Queries

Search queries are essential for our problem setting, as detailed in Section 1.2. Real-world queries for Amazon items are available from the Shopping Queries Dataset [21], which contains queries that real consumers have used, along with relevant products. However, the overlap between that dataset's item space and the review dataset's item space is low.

In addition to those real-world queries, we therefore chose to synthetically generate 10 queries for each item, reserving one at random for validation. We used Llama-3.1-8B-Instruct with a prompt based on the item details such as its title, description, and brand; more details can be found in Section 6.0.1.

We reserve the real-world queries for each item where possible to construct a test set.

2.1.2 Prices

For each purchase, in order to capture price sensitivity, we needed prices at the time of purchase for a set of counterfactual item purchases. We used the Keepa API [22], which offers access to their historical price data for Amazon products.

Since we had the time at which each review was left, but not the exact time of purchase, we had to approximate the time of purchase and match it with our historical prices. To do this, we first extracted the lowest prices over each week over a large time period for each item in our

dataset. This is done to smooth prices, since they can vary largely over a short timeframe. Then, we aligned review times to the immediately preceding week period, using the previously computed lowest prices for items. When the review timestamp is out of the range of our historical prices, we instead used the average lowest weekly price.

During model training and inference, we clip prices to be between the 1% and 99% percentiles of review prices to account for outliers. For instance, around 0.87% of review prices are $0.01, which might be indicative of a pricing error, or that those purchased prices did not keep in track other costs like shipping.

2.2 Dataset Statistics

The resulting dataset encompasses 2,888,322 reviews across three product categories, with significant differences in review count, item count, and average prices as shown in Table 1.

Amazon Category	Users	Items	Avg # Reviews / Item	Avg Purchased Price
Movies and TV	212,282	43,657	53.74	$19.21
Video Games	42,490	12,147	30.11	$39.34
Musical Instruments	21,794	8,074	21.84	$49.32
Combined	261,957	63,878	45.22	$23.60

Table 1: Overall dataset statistics.

Substantial variation exists across the categories. For instance, *Movies and TV* has the lowest average purchased price and the most users, while *Musical Instruments* has the highest average purchased price and the least users.

# Movies and TV & Video Games	# Movies and TV & Musical Instruments	# Musical Instruments & Video Games	# All three
10,959	2,873	1,208	431

Table 2: Number of users who have purchases in multiple categories.

Our dataset also captures users who purchase across multiple categories, with significant overlap between the *Movies and TV* and *Video Games* categories, as detailed in Table 2.

2.2.1 Test-set Statistics

We chose to train our model with synthetic queries and evaluate using real-world queries. To construct the test set, we selected a subset of users whose sequence contains an item which has an associated real-world query (details on this selection process is included in Section 4.1.1).

Amazon Category	Users	Items	Avg # Reviews / Item	Avg Purchased Price
Movies and TV	7,407	575	12.88	$16.85
Video Games	9,104	809	11.25	$51.77
Musical Instruments	4,182	599	6.98	$50.23
Combined	20,693	1,983	10.44	$38.95

Table 3: Test-set statistics. Here, all statistics refer to review-item pairs which are used as evaluatory purchases for our test-set.

# Movies and TV & Video Games	# Movies and TV & Musical Instruments	# Musical Instruments & Video Games	# All three
1,782	426	300	79

Table 4: Number of test-set users who have purchases in multiple categories.

Despite being a subset of our item space, we observe a healthy spread across categories in our test-set, both in overall statistics (Table 3), and in cross-domain users (Table 4).

2.3 Limitations

Our dataset allows for the evaluation of buyer-side recommendation agents with varying prices and contextual user desires. However, several limitations should be acknowledged.

- **Query accuracy**: Both our synthetic and real-world queries may not exactly match what users used when searching for these items. For example, we might assign the real-world query "fruit" to a purchase of apples, based on previous data with that query-item pair. But, the user for the specific purchase we are predicting may have actually searched specifically for apples. This may affect the quality of price sensitivity estimation.
- **Timestamp accuracy**: Using review timestamps as a proxy for purchase times introduces noise. Some users may review a product significantly after a purchase, or may not be strategic or successfully forecast short-term price movements, misaligning the true price with our assigned price data.
- **Selection bias**: The dataset only contains purchases, lacking instances where users searched for an item, and chose not to purchase anything. This absence could cause our price sensitivity estimates to be too low.
- **Platform effect**: It is reasonable to expect consumers to purchase items within the first few pages of their search. Therefore, the purchases in our dataset are potentially biased towards Amazon's recommender system suggestions, where the ideal item may not have appeared.

Chapter 3

Model

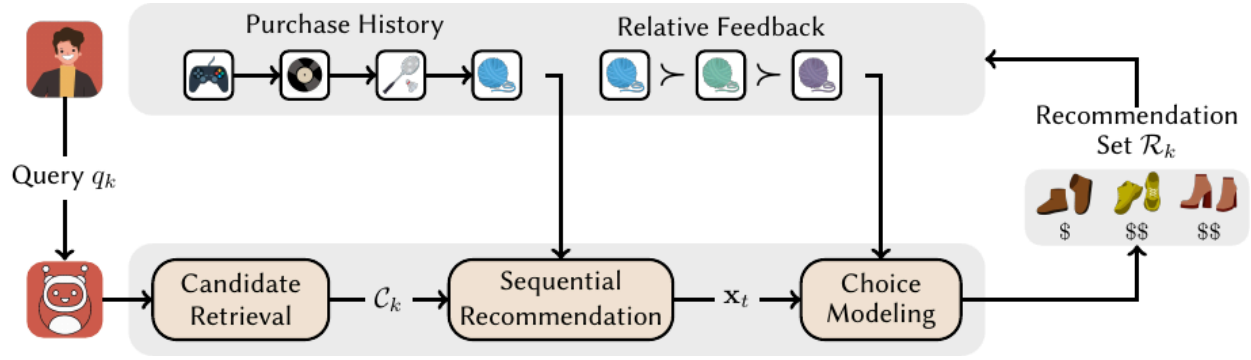


Figure 2: Our Recommendation Agent architecture.

Our recommendation agent architecture consists of several modules. They iteratively narrow the item space to a small set of substitutable items suitable for the consumer's contextual interest, providing personalization in price sensitivity and items during the process. We detail our choices for each module.

3.1 Candidate Retrieval

Our system begins with a retrieval step, which takes a query q_t as input, and retrieves a large candidate set of semantically relevant items $\mathcal{C}_t$.

We use nomic-embed-text-v2-moe [23], a text embedding model, to generate embeddings for both the query and items. Item embeddings are constructed using metadata, such as titles and descriptions. We then use cosine similarity to rank the items, and retrieve the top 100 items for our candidate set. Specific details can be found in Section 6.0.2.

Using a natural language model (as opposed to learning embeddings from scratch over query-input pairs) allows for arbitrary query input. This is necessary for both real-world use, as well as the evaluation of our model on unseen real-world queries.

This component can also be substituted by other retrieval models, which is an area that has previously been well-studied [24], [25], [26].

3.2 Sequential Recommendation

Next, we apply a sequential recommendation model to filter $\mathcal{C}_t$ down to a recommendation set $\mathcal{R}_t$, providing personalization based on previous purchases.

We adopt the transformer architecture used in Query-SeqRec [8], which takes as input a sequence of item-interactions and queries, and tries to predict the next item.

We test both including and excluding queries from the input sequence to the transformer (excluding reduces the model to SASRec [6]). We fix query embeddings to be the text embeddings from candidate retrieval. Because test-time queries are unseen during training, we do not directly fine-tune the query embeddings. Instead, we learn a projection on the query text embeddings. For item embeddings, we follow Harte et al. [27] in using the text embeddings from Section 3.1 as initialization, allowing the model to update them during training. We take the top 20 items ranked by the sequential model for the next step.

Other sequential recommendation models can be substituted in this stage.

3.3 Discrete Choice Modeling

After receiving a small set of substitute items $\mathcal{R}_t$ from the previous step, we now implement price sensitivity through a discrete choice model, which re-ranks the items.

Discrete choice models [19], a popular framework that has seen use in areas such as transportation [28], are an attractive choice for their connection to random utility theory, which allows us to model user utility functions under noise.

We assume that consumer purchase behavior follows the multinomial logit (MNL) model. For some individual consumer, under the MNL model, the utility $U_{i,t}$ of a product i at time t is comprised of two parts: a deterministic component $u_{i,t}(p_{i,t})$, and a random component $\xi_{i,t}$ drawn from a zero-mean Gumbel distribution:

$$U_{i,t} = u_{i,t}(p_{i,t}) + \xi_{i,t}.$$

If the user chooses the highest utility item from a set of recommended items $\mathcal{R}_t$ for query q_t, their choice of item is probabilistic, and follows a softmax over the deterministic utilities:

$$\Pr\big(U_{i,t} \geq \max\{U_{j,t} : j \in \mathcal{R}_t\}\big) = \frac{e^{u_{i,t}(p_{i,t})}}{\sum_{j \in \mathcal{R}_t} e^{u_{j,t}(p_{j,t})}}.$$

As a function of each item's current price $p_{i,t}$ and the user's query q_t, we assume the deterministic utility is given by the following model:

$$u_{i,t}(p_{i,t}) = \underbrace{\langle x_t, y_i \rangle + b_{1,i}}_{\text{sequential recommendation}} \underbrace{-\big(\alpha_{q_t} + \alpha_u\big) f(p_{i,t}) + b_{2,i}}_{\text{choice model price sensitivity}},$$

where x_t is the final hidden-state output of the transformer based on the consumer's sequence history, vector y_i is the learned embedding of item i, biases b_1, b_2 are learned per-item values which can account for hidden factors such as item popularity, and α_{q_t}, α_u are parameters measuring price sensitivity. We argue it is necessary to learn another bias term after sequential recommendation because b_1 may have implicitly captured average item prices, so after pretraining (see Section 4.1.4), another bias b_2 may be necessary to offset b_1's implicit price component to encourage the model to use the real price feature.

This model assumes that utilities vary some transformation of price. We test both log and Box-Cox transformation [29] of price, which sets

$$f(p_{i,t}) = \frac{p_{i,t}^{\lambda_{q_t}} - 1}{\lambda_{q_t}}$$

for some learned parameter λ_{q_t}. The parameter controls the concavity of $f(p_{i,t})$: for instance, $\lambda_{q_t} = 1$ causes $f(p_{i,t})$ to be linear, while $\lambda_{q_t} = 0$ lets $f(p_{i,t}) = \log(p_{i,t})$. Using this transform allows us to test whether the model learns different concavities for different contextual desires.

In order to capture varying price sensitivity across product categories, we compute α_{q_t} and λ_{q_t} as the outputs of multi-layer perceptrons whose input is the query q_t, and with final activation functions softplus and tanh, respectively, to restrict values to a reasonable range.

In this step, α_u is set to 0.

3.4 Incorporating Relative Feedback

After the model is trained on user sequences, we can then adjust price sensitivity for each user based on the user's choice between the recommended items, allowing the model to capture idiosyncratic price sensitivity. In this step we freeze all other parameters besides α_u and ε.

After discrete choice modeling, we have a recommendation set $\mathcal{R}_t$. The optimization problem based on past user purchases over each set is

$$\min_{\alpha_u} \sum_{t\in[T]} \mathcal{L}(\alpha_u) = \max_{\alpha_u} \sum_{t\in[T]} \log \frac{\exp\left(\langle x_t, y_i\rangle + b_{1,i} - \left(\alpha_{q_t} + \alpha_u\right) f(p_{i,t}) + b_{2,i}\right)}{\sum_{j\in\mathcal{R}_t} \exp\left(\langle x_t, y_j\rangle + b_{1,j} - \left(\alpha_{q_t} + \alpha_u\right) f(p_{j,t}) + b_{2,j}\right)},$$

where $\mathcal{L}$ is the negative log-likelihood of the user's past choices over $\mathcal{R}_t$. This is convex in general, so the minima can be straightforwardly computed. However, because user sequences are often short, we can take gradient descents for each purchase to mitigate overfitting:

$$\alpha_{u,t} \leftarrow \alpha_{u,t-1} - \varepsilon \nabla_{\alpha_u} \mathcal{L}(\alpha_{u,t-1}),$$

for tunable step size ε.

3.5 Privacy

Although this architecture is trained over global purchase history data, unlike recommender systems, our recommendation agent can be hosted locally for each user, sending requests to a market server in a synchronous way. After having been trained, our recommendation agent can therefore keep the user's purchase history and feedback private while still updating through relative feedback (it should be noted that this is not perfect privacy — information about a user's past purchases is reflected in which items the agent negotiates for, which is revealed to sellers, for example).

Chapter 4

Experiments

First, we describe our experiment formulation. Then, we conduct ablation experiments to test the necessity of each component in our RecAgent architecture. Next, we examine our specific modules by comparing the performance between different choices. Lastly, we discuss qualitative properties of what our model learned.

4.1 Problem Setting

4.1.1 Input Sequence Construction

For each user u_n, we construct temporally ordered input sequences

$$S_n = [q_1, i_1, q_2, i_2, ..., q_T, i_T],$$

where q_t are queries, and i_t are items purchased. We truncate inputs to have $T \leq 24$ items to fit in our transformer's context window. Queries can either be real-world (if it exists for the corresponding item), a validation query, or chosen randomly from 9 training queries.

We follow the popular practice of applying leave-one-out train-validation-test split (as in [6]): supposing a sequence contains T purchases, we use the first $T-2$ purchases for training, the $(T-1)$st item for validation, and the last for test.

We train and validate only over synthetic queries and test over the real-world queries. Therefore, we truncate sequences to the last purchased item who has a real-world query, if such an item exists after the first 4 purchased items. We then use these sequences for our test set. We lastly truncate purchase histories to be of length $\leq 24 + 3$.

During training, we assign queries randomly from the training synthetic set. For validation, we fix training queries for each of the first $T-2$ purchases, and use the validation synthetic query for the validation purchase. Similarly for test, we only use sequences whose last purchase has a real-world query, and fix training queries for all other purchases.

4.1.2 Target

For subsequences $[q_1, i_1, q_2, i_2, ..., q_k]$, we are interested in predicting i_k over a candidate set $\mathcal{C}_k$, which consists of the label i_k and some negative samples. During training, we draw negative samples uniformly at random from both the entire item set, as well as from the search candidate set for query q_k. We adopt the cross-entropy loss.

4.1.3 Metrics

We evaluate top-k metrics for Hit Ratio (HR@K), the fraction of times the correct item was listed in the top-k, and Normalized Discounted Cumulative Gain (NDCG@K), which further weights each time the correct item is in the top-k by the rank of the item.

During evaluation, for negative samples, we use the full 100 negative samples from candidate retrieval and include the correct item as positive sample, regardless of whether candidate retrieval failed to retrieve it. We use the top 20 items given by sequential recommendation as our recommendation set for discrete choice modeling.

4.1.4 Training Details

We conduct training in three phases. This is because each component of our architecture relies on the previous component, as alluded to in Section 1.2: discrete choice modeling with prices requires sequential recommendation to identify items that the user would have genuinely considered purchasing, and relative feedback requires discrete choice modeling to have already learned some reasonable base price sensitivity.

Pretraining: First, we remove the price-sensitive term in Section 3.3, and only train the sequential model. We use Adam [30] with learning rate 1e-3, $\beta_1 = 0.9, \beta_2 = 0.98$, and apply weight decay 1e-4. Our transformer model is trained with dropout 0.5 and has 4 transformer blocks and 4 attention heads. We train with batch-size 256, and for each positive sample, we use 4 negative samples for training: 2 drawn randomly over the item space, and 2 drawn randomly from the candidate retrieval step (Section 3.1). We found that the exact ratio between the negative samples appears to not make much of a difference.

Post-training: Next, we freeze the sequential recommender, and learn the price-sensitive parameters in Section 3.3, using the same optimizer. Over 20 negative samples drawn from the search candidates, we sample the top 5 from sequential recommendation and add the label for our set of positive and negative samples.

Relative feedback: Lastly, using the same recommendation set sampling procedure as in post-training, we learn idiosyncratic α_u as detailed in Section 3.4, searching the learning rate over validation performance, and with the same sampling procedure as in post-training.

We log validation and test performance every 3 epochs in pretraining and every 5 epochs in post-training. For both, if the validation target fails to beat the previous best validation target twice in HR@5, we end the training run. In relative feedback, we search step size ε over a fixed set, also choosing based on validation HR@5.

4.2 Results

4.2.1 Candidate Retrieval Evaluation

Our setup assumes the candidate retrieval is perfect in retrieving the correct item, since we always include the correct item as a candidate, regardless of whether it's returned by the

search model. We present data on evaluating this assumption by computing the Hit Ratio and NDCG over query-item pairs, using the rest of the entire item set as negative samples.

Query Set	HR@10	HR@100	HR@500	NDCG@10	NDCG@100	NDCG@500
Synthetic	0.3108	0.5207	0.6695	0.2124	0.2551	0.2743
Real-World	0.2235	0.4750	0.6438	0.1349	0.1854	0.2072

Table 5: Candidate retrieval step evaluation. Training and validation queries are synthetically generated, and test queries are real-world.

The results in Table 5 show that real-world queries are more difficult for our retrieval than synthetically generated queries. This is likely due to the fact that the authors of the Shopping Queries Dataset specifically sampled queries with interesting properties.

4.2.2 Recommendation Agent Evaluation

We now evaluate our recommendation agent architecture with an ablation study.

Models	HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20
Cand Ret Only	0.19316	0.23617	0.29976	0.14365	0.15742	0.17325
TextEmbed + Bias	0.70035 ± 0.00307	0.80030 ± 0.00230	0.88023 ± 0.00142	0.55291 ± 0.00383	0.58540 ± 0.00365	0.60572 ± 0.00343
SASRec	0.74147 ± 0.00109	0.83970 ± 0.00219	**0.91100** ± 0.00148	0.59154 ± 0.00222	0.62349 ± 0.00213	0.64168 ± 0.00210
SASRec + MNL	**0.74791** ± 0.00086	**0.84378** ± 0.00136	**0.91100** ± 0.00148	**0.59962** ± 0.00155	**0.63082** ± 0.00127	**0.64797** ± 0.00128
SASRec + MNL + Feedback	0.74786 ± 0.00095	0.84367 ± 0.00147	**0.91100** ± 0.00148	0.59945 ± 0.00170	0.63064 ± 0.00140	0.64781 ± 0.00127

Table 6: Ablation study of the RecAgent components over the test set. Where applicable, the 95% confidence interval is included.

In Table 6, we use the candidate retrieval step for all models and average over 5 random seeds where applicable, using a Student's t-distribution for computing error bars.

- **Cand Ret**: Cosine similarity between query and items embeddings in Section 3.1.
- **TextEmbed + Bias**: Similar to [27]'s LLMSeqSim, we use cosine similarity between the last-purchased item (instead of query) and the candidate items. However, we also add a learned bias onto the logits.
- **SASRec**: SASRec [6] with learned biases and initialized pre-trained text embeddings.
- **SASRec + MNL**: The same as above, but with the discrete choice modeling step.
- **SASRec + MNL + Feedback**: Same as above, but with relative feedback. We chose to compute gradients of the idiosyncratic price sensitivity parameter α_u with respect to the entire sequence instead of updating temporally, since the latter approach failed to elicit good results.

We observe that TextEmbed + Bias is able to achieve strong performance, despite being a simple method. We hypothesize that this is due to our item space having large right-tails in popularity, which the bias term is able to incorporate as a prior, as plotted in Figure 3.

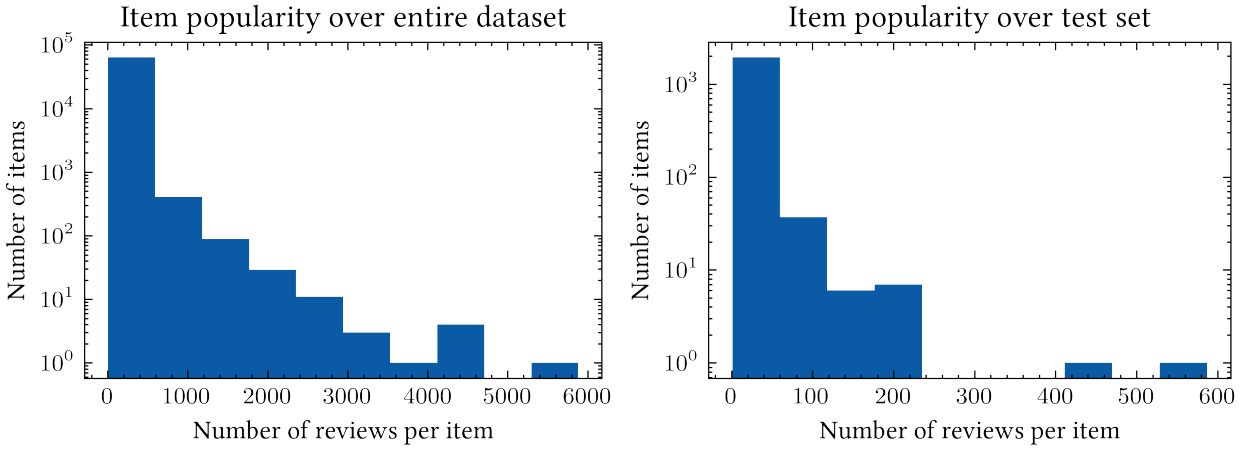


Figure 3: Distribution of item popularity. The right-tail items may allow for good evaluated predictive power from learned biases per-item, which can capture popularity priors.

We find that the relative feedback step appears to make essentially statistically insignificant changes to performance—discussion on this result can be found in Section 4.3. Nonetheless, we observe that the MNL stage, by incorporating price information, is able to make improvements on the sequential recommendation stage, validating the significance of its learned price sensitivities.

4.2.3 Sequential Recommendation Evaluation

We now measure the effect of different sequential modules, ignoring price information.

Model	HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20
Query-SeqRec	0.73740 ± 0.01358	0.83407 ± 0.02022	0.90406 ± 0.01759	0.58215 ± 0.00760	0.61360 ± 0.00816	0.63139 ± 0.00531
Random Embed Init	0.73134 ± 0.00120	0.83205 ± 0.00329	0.90808 ± 0.00094	0.58560 ± 0.00191	0.61838 ± 0.00116	0.63772 ± 0.00150
SASRec	**0.74147** ± 0.00109	**0.83970** ± 0.00219	**0.91100** ± 0.00148	**0.59154** ± 0.00222	**0.62349** ± 0.00213	**0.64168** ± 0.00210

Table 7: Comparison between different sequential modules without price sensitivity. For the first two rows, 3 random seeds are used.

In Table 7, for Query-SeqRec, we initialize both embeddings using pre-trained text embeddings, but freeze query embeddings, since we use a pre-trained text embedding model. To allow for modification of query embeddings, we add a linear layer after query embeddings.

We note high variance in Query-SeqRec's performance. Under our setting, because our queries are synthetically generated and we use a pre-trained embedding model unlike the authors' original work [8], this is possibly due to overfitting to the synthetic distribution.

We are able to replicate Harte et al.'s [27]'s result in our setting: initializing SASRec with pre-trained text embeddings is able to outperform learning embeddings from scratch, whose results are listed under Random Embed Init.

4.2.4 Discrete Choice Modeling Evaluation

Transform	HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20
log Price	**0.74833** ± 0.00085	**0.84442** ± 0.00087	**0.91100** ± 0.00148	**0.59968** ± 0.00170	**0.63093** ± 0.00151	0.64791 ± 0.00153
Box-Cox Price	0.74791 ± 0.00086	0.84378 ± 0.00136	**0.91100** ± 0.00148	0.59962 ± 0.00155	0.63082 ± 0.00127	**0.64797** ± 0.00128

Table 8: Comparison between different price sensitivity functions without relative feedback. 95% confidence intervals are shown.

For the models in Table 6, we choose the Box-Cox transform of price. We now evaluate it against using log transform in Table 8. Here, we are testing it over SASRec + MNL models without relative feedback. We notice little difference in performance between the two. It is possible that this is because $a \log b$ terms may resemble scaled Box-Cox transforms over the values we are analyzing.

4.3 Qualitative Results and Discussion

We now investigate some properties of the trained models.

4.3.1 Price Sensitivity Parameters

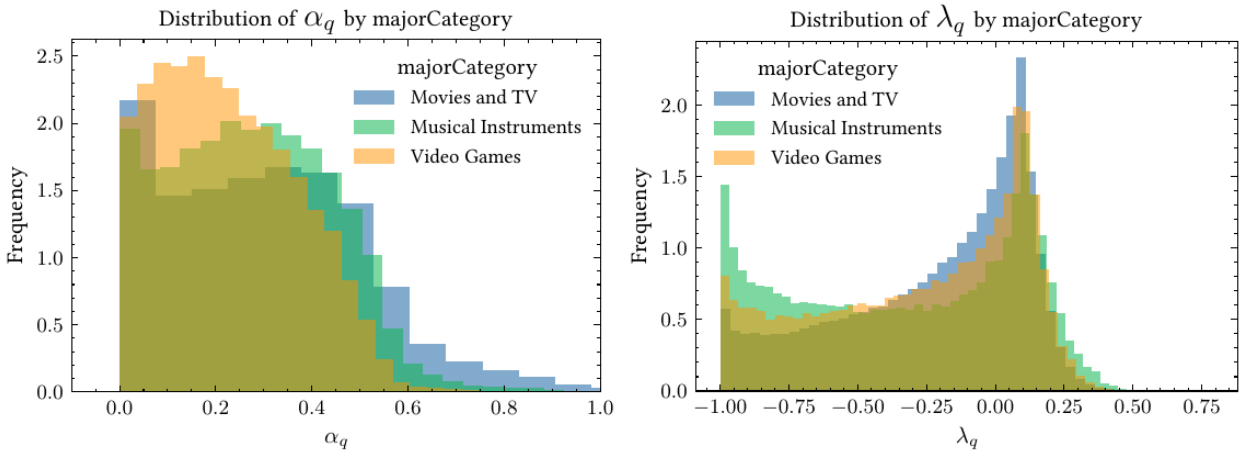


Figure 4: Distribution (in density form) of learned price-relevant parameters, plotted per product category, and over the full query dataset. Differing characteristics between the categories can be observed.

In Figure 4, we plot the price-relevant parameters learned per-category. We observe that *Movies and TV* is the most price-sensitive category, while *Video Games* is the least.

On the other hand, the concavity distribution (given by λ_q) is relatively similar across categories, mostly similar or more concave than log. In particular, *Musical Instruments* tends to

have more queries with heavier concavity—this might be explained by the presence of expensive equipment in the category, for which consumers may be more price-indifferent.

4.3.2 Relative Feedback

We observed that the relative feedback step did not make significant performance gains.

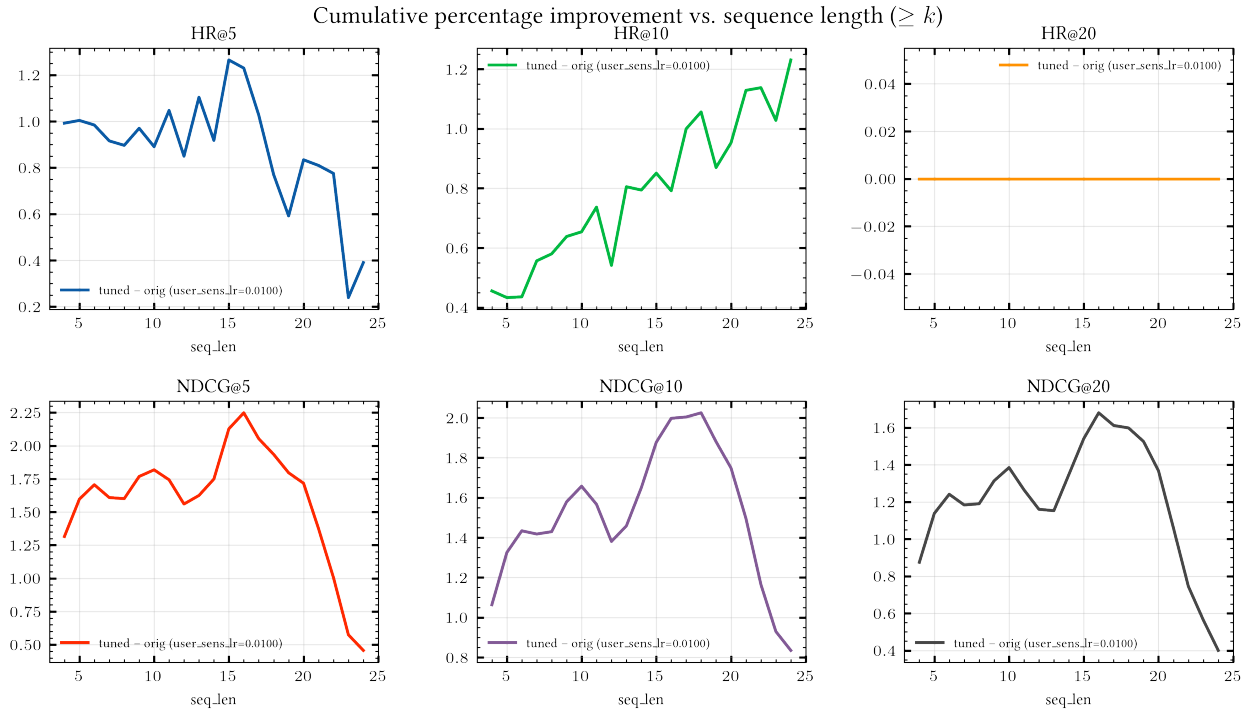


Figure 5: Percentage improvement for each metric of relative feedback over SASRec + MNL, plotted over datapoints of purchase sequence length $\geq k$, and over one pair of trained models. Note that improvement of HR@20 is constant at 0, because both models rely on the same SASRec model to identify the top 20 recommended items.

Here, we investigate where improvements, and lack thereof, come from. We might expect that if relative feedback was properly capturing predictive trends, that it would make larger improvements over the SASRec + MNL model when user purchase sequences are longer, since that gives more data to learn idiosyncratic price sensitivity.

In Figure 5, we observe that for one training run where relative feedback did improve the model, HR@10 follows this trend, steadily improving as a user's sequence length increases. However, improvements in all other metrics peak somewhere in the middle, then decreases, hinting at some issue with our gradient updating method.

Relative feedback may not have worked because our gradient approximation method is simply too inaccurate for our setting, in which the data of a few user price purchases is too noisy. It is also possible that our dataset's products are not very substitutable, causing later steps in our architecture, which are finer-grained and rely on substitutability, to face challenges in capturing significant trends. We leave the investigation of the application of more robust

learning methods and the exploration of other product categories with more substitutable goods for future work—more discussion can be found in Section 5.2.

Chapter 5

Conclusion and Future Work

We outlined the implementation of recommendation agent systems, which are systems that approximate the utility function of a single buyer, and that therefore can negotiate on the buyer's behalf. Using a semi-synthetic dataset, we evaluated our RecAgent architecture's ability to capture predictive user preference and price sensitivity.

5.1 Market Dynamics

Our agents model consumer decisions as probabilistic. Under this assumption, for some fixed price vector, purchase probabilities for each consumer is computable—therefore, a seller's revenue is also probabilistic, whose distribution may be calculated.

5.1.1 Price Learning

Under this setting, because the feedback from consumer demand is noisy, seller pricing algorithms must balance exploration (setting lower prices to get feedback on consumer demand) and exploitation (setting higher prices which may give higher revenue, but provides less signal). We therefore propose that work such as Erginbas et al.'s [18], which studies the joint problem of recommendation and pricing from a seller's perspective when users follow the MNL model, or Weed et al.'s [31], which applies bandit learning in sequential auctions, may serve as a seller-side counterpart to RecAgent, allowing for a real-world approximation of a setting similar to Greenwald and Kephart's shopbots and pricebots [17].

5.1.2 Price Competition

The equilibria of our market setting may be analyzed with past work such as Morrow and Skerlos's analysis of the existence of Bertrand-Nash equilibrium prices under logit demand [32]. Since sellers can consider their revenue distributionally, our market setting might also relate to Li and Webster's study of logit demand markets with risk-sensitive sellers [33].

5.2 Future Work

Dataset: We chose three Amazon categories to construct our dataset. Other categories and platforms may be interesting to study. For instance, food items are likely to be more substitutable than movies. Also, platforms like Uber may have more price variability, which could allow our RecAgent architecture to see greater improvements over non-price-sensitive baselines.

Discrete choice modeling: Other terms may be explicitly incorporated in a linear fashion into the discrete choice modeling in Section 3.3. For instance, users may have different sensitivities to the average rating of a product, whose information is available in Ni et al.'s dataset [20].

Relative feedback: The rating of the user, in addition to the sentiment given by their review, may be used as more fine-grained feedback. A user might also rank the recommended items provided to them, as depicted in Figure 2. Furthermore, we found the optimization of parameter α_u to be tricky over our sequence sizes (≤ 27)—it would be interesting to study the application of more robust optimization methods for this step.

Two-sided market dynamics: It also remains to see how our buyer recommendation agents and price-learning seller systems interact. It could be interesting to study empirically how efficient the prices they converge to are, as well as theoretically how fast, if at all, their interactions converge towards equilibria.

Chapter 6

Appendix

6.0.1 Synthetic Query Generation

To generate synthetic queries, we use the sampling parameters `temperature=1.0`, `top_p=0.95`, and the following prompt for Llama-3.1-8B-Instruct:

```
"""You are a search query generator for an online marketplace. Your task is to create realistic search
queries that potential customers would use when looking for specific products.

INPUT FORMAT:
You will receive product information in the following fields:
- TITLE: The full product title
- DESCRIPTION: Detailed product description
- BRAND: Product brand name (may be missing)
- CATEGORY: Product category (may be missing)
- OTHER CATEGORIES: Other relevant categories (may be missing)

OUTPUT REQUIREMENTS:
- Generate exactly 10 different search queries separated by commas
- Each query should be less than 5 words long
- Include a mix of specific and general search terms
- Output ONLY the comma-separated queries with no additional text

QUERY GUIDELINES:
- Create queries that real users would type when searching for this product
- Include important product features, functionality, and use cases
- Vary between specific (with brand names) and generic queries (across the class of products)
- Use common marketplace search terminology and patterns
- Avoid overly technical specifications unless they're search-relevant
- Don't mention inconsistencies in the provided information

EXAMPLES:

Example:
TITLE: Yamaha CGS103A 3/4-Size Classical Guitar Bundle with Gig Bag, Tuner, Strings, String Winder, Austin
Bazaar Instructional DVD, and Polishing Cloth
DESCRIPTION: Adding value to your purchase, Austin Bazaar bundles your instrument with necessary
accessories. Everything you need to start playing immediately comes in one box. Save yourself the hassle
and save some money while you're at it. A gig bag is included so you can keep your instrument safely packed
away
BRAND: Yamaha
CATEGORY: Musical Instruments
OTHER CATEGORIES: Musical Instruments, Guitars, Classical & Nylon-String Guitars

Output:
yamaha classical guitar bundle, 3/4 size classical guitar, yamaha cgs103a beginner guitar, classical guitar
starter kit, classical guitar yamaha, 3/4 guitar with accessories, yamaha guitar with gig bag, classical
guitar tuner bundle, yamaha guitar starter pack, beginner nylon string guitar
"""
```

6.0.2 Candidate Retrieval

We use nomic-embed-text-v2-moe, embed queries using the “query” option, and items using the “passage” option, using the title, description, brand, major category, and minor categories

as text. Specifically, we format item inputs as `"TITLE DESCRIPTION Brand: BRAND | Category: MAJOR_CATEGORY | Tags: MINOR_CATEGORIES"`.

We truncate the embeddings to dimension 64, utilizing the text embedding model's Matryoshka Embedding property [34].

Bibliography

[1] P. Ferreira, X. Zhang, R. Belo, and M. Godinho de Matos, "Welfare properties of recommender systems: Theory and results from a randomized experiment," *Available at SSRN 2856794*, 2016.

[2] K. Arrow, "J.(1951),"An extension of the basic theorems of classical welfare economics,," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, J. Neyman, ed., University of California Press, Berkeley, CA*,

[3] G. Debreu, "The coefficient of resource utilization," *Econometrica: Journal of the Econometric Society*, pp. 273–292, 1951.

[4] K. J. Arrow and G. Debreu, "Existence of an Equilibrium for a Competitive Economy," *Econometrica*, vol. 22, no. 3, pp. 265–290, 1954, Accessed: May 08, 2025. [Online]. Available: http://www.jstor.org/stable/1907353

[5] L. T. Liu, H. Mania, and M. Jordan, "Competing bandits in matching markets," in *International Conference on Artificial Intelligence and Statistics*, 2020, pp. 1618–1628.

[6] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE international conference on data mining (ICDM)*, 2018, pp. 197–206.

[7] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] Z. He, H. Zhao, Z. Wang, Z. Lin, A. Kale, and J. Mcauley, "Query-aware sequential recommendation," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 4019–4023.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[10] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[11] F. Sun *et al.*, "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.

[12] C. Liu, J. Lin, J. Wang, H. Liu, and J. Caverlee, "Mamba4Rec: Towards Efficient Sequential Recommendation with Selective State Space Models." [Online]. Available: https://arxiv.org/abs/2403.03900

[13] J. Wang and Y. Zhang, "Utilizing marginal net utility for recommendation in e-commerce," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, 2011, pp. 1003–1012.

[14] F. J. Ruiz, S. Athey, and D. M. Blei, "Shopper," *The Annals of Applied Statistics*, vol. 14, no. 1, pp. 1–27, 2020.

[15] Y. Zheng, C. Gao, X. He, Y. Li, and D. Jin, "Price-aware recommendation with graph convolutional networks," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 133–144.

[16] M. Wan *et al.*, "Modeling consumer preferences and price sensitivities from large-scale grocery shopping transaction logs," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 1103–1112.

[17] A. R. Greenwald and J. O. Kephart, "Shopbots and pricebots," in *Agent Mediated Electronic Commerce II: Towards Next-Generation Agent-Based Electronic Commerce Systems 2*, 2000, pp. 1–23.

[18] Y. E. Erginbas, T. A. Courtade, and K. Ramchandran, "Online Assortment and Price Optimization Under Contextual Choice Models." [Online]. Available: https://arxiv.org/abs/2503.11819

[19] D. McFadden, "Conditional logit analysis of qualitative choice behavior," 1972.

[20] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.

[21] C. K. Reddy *et al.*, "Shopping Queries Dataset: A Large-Scale ESCI Benchmark for Improving Product Search," 2022.

[22] Keepa, "Keepa API." Accessed: May 15, 2025. [Online]. Available: https://keepa.com/#!api

[23] Z. Nussbaum and B. Duderstadt, "Training Sparse Mixture Of Experts Text Embedding Models," *arXiv preprint arXiv:2502.07972*, 2025.

[24] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 191–198.

[25] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[26] J.-T. Huang *et al.*, "Embedding-based retrieval in facebook search," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2553–2561.

[27] J. Harte, W. Zorgdrager, P. Louridas, A. Katsifodimos, D. Jannach, and M. Fragkoulis, "Leveraging Large Language Models for Sequential Recommendation," in *Proceedings of the 17th ACM Conference on Recommender Systems*, in RecSys '23. ACM, Sep. 2023, pp. 1096–1102. doi: 10.1145/3604915.3610639.

[28] M. E. Ben-Akiva and S. R. Lerman, *Discrete choice analysis: theory and application to travel demand*, vol. 9. MIT press, 1985.

[29] G. E. P. Box and D. R. Cox, "An Analysis of Transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 26, no. 2, pp. 211–252, 1964, Accessed: May 08, 2025. [Online]. Available: http://www.jstor.org/stable/2984418

[30] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] J. Weed, V. Perchet, and P. Rigollet, "Online learning in repeated auctions," in *Conference on Learning Theory*, 2016, pp. 1562–1583.

[32] W. R. Morrow and S. J. Skerlos, "On the Existence of Bertrand-Nash Equilibrium Prices Under Logit Demand." [Online]. Available: https://arxiv.org/abs/1012.5832

[33] H. Li and S. Webster, "Risk sensitivity and firm power: Price competition with mean2-variance profit objective under multinomial logit demand," *Operations Research*, vol. 72, no. 3, pp. 957–965, 2024.

[34] A. Kusupati *et al.*, "Matryoshka representation learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 30233–30249, 2022.