Augmenting Guest Search Results with Recommendations at Airbnb

Haowei Zhang
haowei.zhang@airbnb.com
Airbnb Inc
San Francisco, CA, USA

Philbert Lin
philbert.lin@airbnb.com
Airbnb Inc

Dishant Ailawadi
dishant.ailawadi@airbnb.com
Airbnb Inc

Soumyadip Banerjee
soumyadip.banerjee@airbnb.com
Airbnb Inc

Shashank Dabriwal
shashank.dabriwal@airbnb.com
Airbnb Inc

Hao Li
hao.li@airbnb.com
Airbnb Inc

Kedar Bellare
kedar.bellare@airbnb.com
Airbnb Inc

Liwei He
liwei.he@airbnb.com
Airbnb Inc

Sanjeev Katariya
sanjeev.katariya@airbnb.com
Airbnb Inc
San Francisco, CA, USA

Abstract

Airbnb operates as a unique two-sided marketplace, connecting hosts offering distinctive accommodations with guests possessing diverse travel needs and preferences. Guests often engage in exhaustive searches using varying conditions to find suitable listings. However, overly narrow search criteria can result in insufficient results, causing frustration and abandonment of search journeys. To address these challenges, we enhanced Airbnb's search experience by augmenting results with flexible date recommendations and relaxed amenity and price filters, aligning with guests' broader travel intent. This approach has significantly improved guest bookings on the platform.

Key innovations include: (1) a modular and extensible recommendation architecture that leverages Airbnb's existing search ranking system, ensuring scalability, rapid iteration, and maintainability; and (2) an efficient solution through transfer learning and a Mixture of Experts (MoE) architecture to accurately rank recommendation carousels alongside organic search results. This solution seamlessly handles scenarios ranging from single to multiple recommendations applicable to various search conditions, addressing cold start challenges while enabling continuous iterations. Our robust, scalable, and extensible solution is not only tailored for Airbnb but is also applicable to other industries facing similar challenges—such as online travel agencies and e-commerce platforms—where users require recommendations for infrequent but high-stakes decisions.

CCS Concepts

• **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Keywords

machine learning, recommendation, e-commerce, learning to rank

ACM Reference Format:
Haowei Zhang, Philbert Lin, Dishant Ailawadi, Soumyadip Banerjee, Shashank Dabriwal, Hao Li, Kedar Bellare, Liwei He, and Sanjeev Katariya. 2025. Augmenting Guest Search Results with Recommendations at Airbnb. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference '25).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Airbnb operates as a dynamic two-sided marketplace, connecting hosts offering unique accommodations with guests seeking distinctive travel experiences worldwide [5, 8]. Each Airbnb listing is characterized by specific features–such as location, amenities, pricing, and varying check-in/check-out policies – that distinguish it from traditional accommodations, presenting both opportunities and challenges for hosts and guests alike [3, 4]. Simultaneously, guests arrive on the platform with diverse needs and preferences, ranging from a desire for a tranquil retreat to an adventure-filled stay or a workspace-friendly environment [4]. This interplay of individuality results in an inherently complex search experience for guests. Similar to other online travel booking platforms, guests often conduct tens or even hundreds of searches over several days or weeks before finalizing a booking [10]. This exhaustive process stems from the high-stakes nature of travel accommodation decisions, where each booking typically represents a significant financial commitment – often ranging from hundreds to thousands of dollars—and constitutes a major portion of the overall travel budget.

For Airbnb guests, these challenges are amplified by the platform's unique marketplace dynamics. During peak travel seasons, inventory often depletes quickly, making it difficult for guests planning trips with limited lead times. The distinct features of each listing—such as amenities, pricing, and policies—prompt guests to apply diverse search criteria, which can result in overly narrow searches and limited available options. Frustration from insufficient results may lead guests to abandon their search prematurely.

When guests searches yield insufficient results, it is essential to recommend results align with their broader travel intentions to assist guests in finding listings. Figure 1 illustrates the user interface of our solution. Since the initial launch in September 2023, the search recommendations has significantly boosted guest bookings.

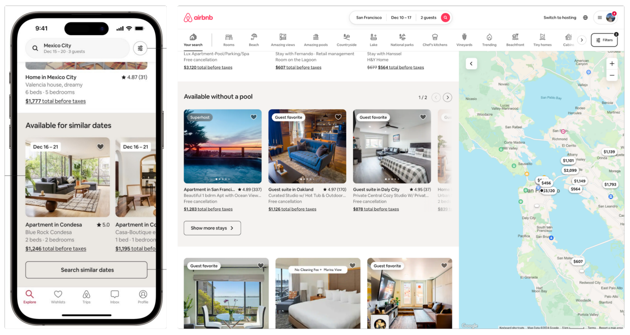

Figure 1: UI of search recommendations, on mobile (left), and desktop (right).

Constructing the aforementioned solution within the context of highly distributed, large-scale online applications like Airbnb's search system presents a range of significant practical challenges. In the next sections we will explain the detail of our solutions to overcoming these challenges. We aim to demonstrate that our solution is not only effective within the context of Airbnb but also has broad applicability to addressing similar challenges faced by other platforms. Specifically, the approach of augmenting user search results by recommending items that align with users' inferred broader intent can be highly beneficial for industries such as Online Travel Agencies and E-Commerce.

2 Architecture of Search Recommendations

Airbnb's unique marketplace characteristics present several practical challenges in building an effective recommendation solution, more specifically:

- Guests use Airbnb infrequently and for high-stakes decisions, with travel intent potentially varying significantly between trips, making accurate recommendations both challenging and crucial.
- Fast response times are vital to maintain a positive guest experience when augmenting search results with recommendations.
- Recommendations must be ranked carefully alongside original search results to enhance the guest experience without causing distraction or confusion.

These factors necessitate real-time recommendations to adapt to guests' evolving needs. The recommendation results must also be generated within the response time limit used by the search system. Before presenting our solution, we first outline the capabilities and constraints of Airbnb's current search ranking system.

2.1 Understanding Airbnb's Search Ranking System

Airbnb's search ranking system is a highly distributed, sharded infrastructure consisting of root nodes and sharded leaf nodes. The process of a search comprises the following steps:

Step 1 - Query Understanding: User queries, such as "San Francisco, July 1st - July 6th, 2 guests," are parsed and converted into Lucene [1] search queries in the root node.

Step 2 - Retrieval: Lucene search are executed on sharded leaf nodes for geo search, amenities, pricing and availability filtering, with each shard storing $1/M$ of the total listings. Up to 10,000 listings are retrieved across all shards.

Step 3 - First-Pass Ranking: n each leaf node, a Learn-to-Rank model [9] uses hundreds of features from User, Query, and Listing to score listings, returning the top 100 to the root node. The model, trained with pairwise cross-entropy loss on booked/non-booked pairs, is optimized for bookings..

Step 4 - Second-Pass Ranking: A root node collects results from all leaf shards and performs final ranking, optimizing for bookings and other business objectives. However, most listing-level features are unavailable at the root node.

2.2 Architecture for Augmenting Search Results with Recommendations

At a high level, generating recommendation items and ranking them involves two primary choices:

(1) Separating the candidate generation system and the recommendation item ranking model.
(2) Executing recommendation query through the existing search ranking system to generate recommendations.

Additionally, ranking across recommendations and the original search results requires the development this new functionality regardless of the choices above.

Compared to the separated system, we decided to build the recommendation capabilities by reusing the existing system. The primary advantage is the ability to treat the recommendation system as modular and extensible. Reusing the capabilities offered by the existing system, we can focus on adding or enhancing functionalities specific to various types of recommendations. This approach saves a significant amount of time and effort to develop and maintain an end-to-end recommendation product and facilitates rapid iteration to introduce new recommendation types or improve existing ones.

Figure 2 illustrates overall recommendation architecture with new components highlighted in the architecture diagram. Section 3 and 4 explain two types of recommendations generation and ranking, while Section 5 details of the new functionality for ranking recommendations alongside original results.

3 Flexible Date Recommendation

Flexible date recommendations provide listings available in the date range of check_in $\pm \delta$ to check_out $\pm \delta$, where δ represents the flexible days range. The key to generate recommendation queries is to determine the optimal δ: A wider δ increases the chances of

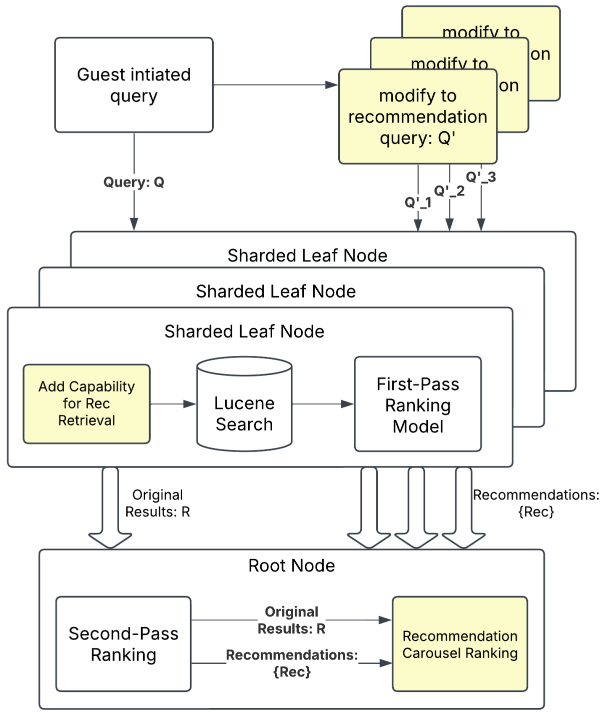

Figure 2: The system architecture for search recommendations on the airbnb search system.

finding listings with better prices and higher quality for guests with flexible travel plans, but an overly broad δ may present less-relevant results and overwhelm guests. De-identified data analysis revealed guest flexibility correlates with trip duration—longer stays often results in wider days range flexibility. Using this insight, the recommendation query is formulated as $\delta = F(\text{total\_stay\_nights})$.

The retrieval and ranking of flexible date recommendations leverage existing capabilities and models in the search ranking system. These same capabilities and models also support guest directly specifying δ through the search bar UI. For each listing $L_i \in \mathbf{Rec}$, where **Rec** represents the recommendation results, the ranking of the recommendation is determined as:

$$\text{Score}(L_i) = f(U, Q', L_i)$$

where U represents user features, Q' represents features of the generated recommendation query, and L_i represents listing features, and f is the first pass ranking model described in section 2.1.

4 Relaxed Filters Recommendations

When guests apply amenities or price filters, relaxed filter recommendations are used to augment results through relaxing non-essential amenity filters or increase price upper bound by 20%. To identify the best amenity filter to relax, we reuse a model that predicts the probability of booking with applied filters Φ in the query $Q(\Phi)$: $P_{\text{book}}(U, Q(\Phi))$. For each relaxable filter $\phi \in \Phi$, we find ϕ that increases the probability most when removed:

$$\underset{\phi}{\text{argmax}}(P_{\text{book}}(U, Q(\Phi \setminus \phi)))$$

The chosen ϕ is then applied to relaxed filter recommendation query.

New retrieval functionality is needed for relax filter recommendations, which was not directly accessible to guests. It is implemented using Lucene search by computing each listing's *filter-distance* from the search request and selecting listings within a specified threshold. If the guest search applies Φ relaxable filters and a listing satisfies $\Psi \subseteq \Phi$, the filter-distance is defined as:

$$\text{Filter-Distance} = |\Phi| - |\Psi|$$

Currently, listings with a filter-distance of 1 or less are selected.

Since guests cannot directly access relaxed filter search features, the production ranking model f lacks training data for relaxed filter listings. As a result, the listing level features in f do not account for relaxed amenities or prices exceeding the applied price filter. Despite this, we use the existing production ranking model as an initial version. For $L_i \in \mathbf{Rec}_{Relax\_Filters}$, the score is computed as:

$$\text{Score}(L_i) = f(U, Q, L_i)$$

where Q represents features of the guest-initiated query, and L_i represents listing-level features, excluding relaxed filter information. This limitation will be addressed in section 5.1.4.

5 Recommendation Carousel Ranking

After removing duplicates from user-initiated search results, top recommendation results are placed in carousels within the search feed. Like typical search engines, user attention drops with lower-ranked content. If the carousel appears too low, even highly relevant recommendations may go unnoticed. Conversely, placing it too high can distract users if the recommendations are not relevant, leading to search abandonment.

Therefore, finding the optimal carousel position is crucial to maximize bookings. We frame the problem as: given a user (U) and search (Q), its results ($\mathcal{R}$), and recommendations (**Rec**), how should carousels be ranked to drive the most bookings? The next sections cover both single and multiple carousel ranking strategies.

5.1 Single Carousel ranking

To maximize guest bookings for a single type of recommendation, the optimal carousel position can be determined by comparing $P_{\text{book}}(\mathbf{Rec})$ against $P_{\text{book}}(L_i)$ for each $L_i \in \mathcal{R}$, as outlined in Algorithm 1:

5.1.1 Practical Considerations. Having described the algorithm to determine the carousel insertion position, we now delve into the practical considerations and challenges associated with implementing this solution within Airbnb's search ranking system.

Representing the Carousel. Without horizontal scrolling, only one listing is visible on mobile devices, and 2–3 listings are visible on the desktop depending on screen resolution. To simplify the problem, we approximate $P_{\text{book}}(\mathbf{Rec})$ by its top-ranked listing booking probability $P_{\text{book}}(\mathbf{Rec}[0])$ in Algorithm 1.

Algorithm 1 Insert Carousel in the Original Result Sequence

```
**Rec** : The ranked recommendation listings in carousel
$\mathcal{R}$ : The ranked original search result listings
$\mathcal{H}$ : Maximum comparison limit
Steps:
compute $P_{\text{Book}}(\mathbf{Rec})$
for enumerate $L_i \in \mathcal{R}[:\mathcal{H}]$ do
    if $P_{\text{book}}(\mathbf{Rec})/P_{\text{book}}(L_i)$ > threshold then:
        return $i$ as the carousel insert position
    end if
end for
```

Feature Availability Limitation. Carousel ranking comparisons between $P_{\text{book}}(\mathbf{Rec}[0])$ and $P_{\text{book}}(L_i), L_i \in \mathcal{R}$ must occur at the root node as shown in Fig 2. However, challenges arise from the lack of listing-level features at the root node as explained in section 2.1.

5.1.2 Addressing Limitations Through Transfer Learning. 5.1.1 simplifies carousel ranking as comparing $P_{\text{book}}(\mathbf{Rec}[0])$ with $P_{\text{book}}(L_i)$. This reduces the problem to a listing-to-listing comparison, motivating us to leverage the first-pass ranking results described in section 2.1.

However, further reformulation is needed to address the following challenges:

(1) **Rec** and $\mathcal{R}$ may originate from different queries, making scores under different query features in the Learn-to-Rank model incomparable [2].
(2) The prediction from the first-pass model can be inaccurate when some key features from **Rec** are missing, as discussed in section 4.

5.1.3 Recommendation Scored Under Different Query Features. In the flexible date recommendation case, as described in section 3, the first-pass score for **Rec** and $\mathcal{R}$ use query features derived from the recommendation query Q' and guest-initiated query Q, respectively. Directly using first-pass score diff under different query features can cause large prediction error. However, we can solve it using the following approach:

If we have $\mathbf{Rec}[0]$ and $L_i \in \mathcal{R}$ both scored under Q, we can use the following formula to compute $P_{\text{book}}(\mathbf{Rec}[0])/P_{\text{book}}(L_i)$ in algorithm 1

$$\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q)}{P_{\text{book}}(L_i, U, Q)} \tag{1}$$

Given that flexible date recommendations retrieves listings available check_in $\pm\ \delta$ to check_out $\pm\ \delta$, which include results with $\delta = 0$, in our system, $L_i \in \mathcal{R}$ are also scored under Q'. We can reformulate (1) as follows:

$$\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q')}{P_{\text{book}}(L_i, U, Q')} \cdot L(U, Q, Q', \mathcal{R}) \tag{2}$$

In formula (2), $L(U, Q, Q', \mathcal{R})$ represents the likelihood of a guest considering switching from the initial query Q and results $\mathcal{R}$ to do the recommendation query Q'. The term $\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q')}{P_{\text{book}}(L_i, U, Q')}$ calculates the relative booking probability in the context of guest switched to query Q'.

By applying log transformation to formula (2), we obtain:

$$log(\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q')}{P_{\text{book}}(L_i, U, Q')}) + log(L(U, Q, Q', \mathcal{R})) \tag{3}$$

Given our first-pass ranking model is trained with a pairwise cross-entropy loss[5, 9], where for the booked listings are labeled as positive and non-booked listings as negative, we can compute $log(\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q')}{P_{\text{book}}(L_i, U, Q')})$ using as the first-pass score difference[4]. Formula (3) becomes:

$$\text{Score}(\mathbf{Rec}[0], U, Q') - \text{Score}(L_i, U, Q') + log(L(U, Q, Q', \mathcal{R})) \tag{4}$$

Therefore, the comparison of $P_{\text{book}}(\mathbf{Rec}[0])$ with $P_{\text{book}}(L_i)$ can be implemented using a trained neural network to represent formula (4).

For the training data, we can leverage features with first-pass score diff of **Rec** and $\mathcal{R}$, user features, query features, and additional features extracted from the original result set $\mathcal{R}$. For the training label, if a guest books on the recommendation listing, the label is positive, if the booking is on organic listing, the label is negative. We refer to this approach as **Learn to Rank Carousel**.

5.1.4 Addressing First-Pass Score Inaccuracy in Recommendations. In the relaxed filters recommendation described in section 4, although the first-pass score of **Rec** and $\mathcal{R}$ is computed under the same query Q, some essential features—such as the relaxed filters ϕ—are missing when scoring **Rec**. However, this issue can also be addressed similarly to the approach as described in 5.1.3, starting with the following formula:

$$\frac{P_{\text{book}}(\mathbf{Rec}[0])}{P_{\text{book}}(L_i)} = \frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q) \cdot P(U, Q, \phi, \mathcal{R})}{P_{\text{book}}(L_i, U, Q)} \tag{5}$$

In formula 5, $P_{\text{book}}(\mathbf{Rec}[0], U, Q)$ estimates the booking probability of $\mathbf{Rec}[0]$ as if it satisfies all the filters in Q. Meanwhile, $P(U, Q, \phi, \mathcal{R})$ corrects the probability of guests (U) considering the relaxed filter ϕ with original search (Q), and results ($\mathcal{R}$).

By applying log transformation on formula 5, we obtain:

$$log(\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q)}{P_{\text{book}}(L_i, U, Q)}) + log(P(U, Q, \phi, \mathcal{R})) \tag{6}$$

This transformed formula 6 closely resembles the earlier formula 3. For the term $log(\frac{P_{\text{book}}(\mathbf{Rec}[0], U, Q)}{P_{\text{book}}(L_i, U, Q)})$, we can leverage the first-pass ranking score difference to compute it, resulting in:

$$\text{Score}(\mathbf{Rec}[0], U, Q) - \text{Score}(L_i, U, Q) + log(P(U, Q, \phi, \mathcal{R})) \tag{7}$$

Similar to formula (4), this formula (7) can rely on the same neural network architecture and a similar feature set for training. This demonstrates the generalizability of our method in addressing both scenarios: when the first-pass score of the recommendation under the different query features and when the first-pass score is inaccurate due to missing features.

5.1.5 Addressing the Cold Start Issue. The model based carousel ranking still faces the the challenge of the lack of training data due to cold start. To address it, we adopted a phased approach to incrementally improve the recommendation system while ensuring a high-quality guest experience as follows:

Initial Version - Conservative Rule-Based Approach: In the initial phase, we compare $P_{\text{book}}(\mathbf{Rec}[0])/P_{\text{book}}(L_i)$ in Algorithm 1 only leveraging the first pass ranking score:

$$\text{Score}(\mathbf{Rec}[0]) - \text{Score}(L_i) > \alpha$$

Here, α was set high to prioritize valuable results and avoid disrupting the search experience. A successful initial version allows us to separate ranking improvements from UI and infrastructure iterations.

Relaxing Thresholds to Enhance Training Data: We reduced α to display recommendations in more scenarios. This is a crucial step to generate more training data for the subsequent model development phase.

Model-Based Carousel Ranking: Based on the training data accumulated from earlier phases, we trained a neural network to predict $P_{\text{book}}(\mathbf{Rec}[0])/P_{\text{book}}(L_i)$, leveraging the formulas 4 and 7. We use Mixture of Experts (MoE) architecture [7] for carousel ranking model, as shown in Fig 3, where we train each sub-network on searches contains corresponding recommendations. This architecture allows us to easily incorporate new carousels, which we will explain more in section 5.2

The model incorporated features such as:

- First-pass score differences between $\mathbf{Rec}[0]$ and $L_i \in \mathcal{R}$.
- User features (e.g., search history, booking preferences).
- Query features from guests initiated (e.g., total nights, lead days, guest numbers, filters applied) and from recommendation (e.g. suggested flexibility).
- Contextual features extracted from the organic result set $\mathcal{R}$ (e.g., total results count).

The results of this phased approach are discussed in detail in Section 6.1.

5.1.6 Modularization between Carousel Ranking and First-Pass Ranking. It is important to note that the carousel ranking model is agnostic to specific first-pass ranking model version. Our first-pass model undergoes continuous improvement, with multiple versions often active in experiments. However, as long as each first-pass model is trained with a cross-entropy loss on booked and non-booked listing pairs, the score difference between two listings can approximate the relative booking probability ratio for a given query. Therefore, the formula (4) and (7) hold across different versions of the first-pass ranking model.

This modularized design results in a clear decoupling between the carousel ranking and the first-pass Ranking. It allows both the first-pass ranking model and the carousel ranking model to iterate and improve independently without concerns about interaction or compatibility issues.

5.2 Multi Recommendation Carousel Ranking

We designed the heuristic Algorithm 2 to determine ranking positions for multiple recommendations. It first calculates the position of each recommendation carousel with respect to $\mathcal{R}$ using either model-based or rule-based approach. It then sort and merge the results to ensure at least N listings between any two carousels in order to avoid distracting guests due to recommendations. Given model-based approach is more accurate in comparing $\mathbf{Rec}$ vs $\mathcal{R}$, in the sort-merge we use high priority queue for carousel C ranked by models, and low priority for C' ranked by rules.

Algorithm 2 Insert Multiple Carousels with Priority Queues

```
{$\mathbf{Rec}_t$}, $t \in \mathcal{T}$ : Ranked recommendations for $\mathcal{T}$ types
$\mathcal{R}$ : Ranked original search results
$N$ : Min gap between carousel positions
$\mathcal{H}_{\text{Top}}$ : Top position for first carousel (e.g., 6)
Steps to Compute Individual Carousel Positions:
for $t \leftarrow 1$ to $T$ do
    Compute $\mathbf{Pos}[t]$ for $\mathbf{Rec}_t$ using Algorithm 1.
end for
$\mathbf{Pos\_sorted} \leftarrow \text{sort}([\mathbf{Pos}[1], \ldots, \mathbf{Pos}[T]])$
Split Pos_sorted into two queues for Multi-Carousel Merging:
  $\mathbf{Pos}_{\text{High\_Pri}}[t]$: Higher Priority, e.g. rank with model, up to 1st page
  $\mathbf{Pos}_{\text{Low\_Pri}}[t]$: Lower Priority, e.g. rank by rule
Steps to Merge Final Positions:
$\mathbf{Pos\_final}[0] \leftarrow \max(\mathbf{Pos}_{\text{High\_Pri}}[0], \mathcal{H}_{\text{Top}})$
for $pos$ : $\mathbf{Pos}_{\text{High\_Pri}}[1:]$ do
    Append $\max(\mathbf{Pos\_final}[-1] + N, pos)$ to Pos_final
end for
for $pos$ : $\mathbf{Pos}_{\text{Low\_Pri}}$ do
    Append $\max(\mathbf{Pos\_final}[-1] + N, pos)$ to Pos_final
end for
```

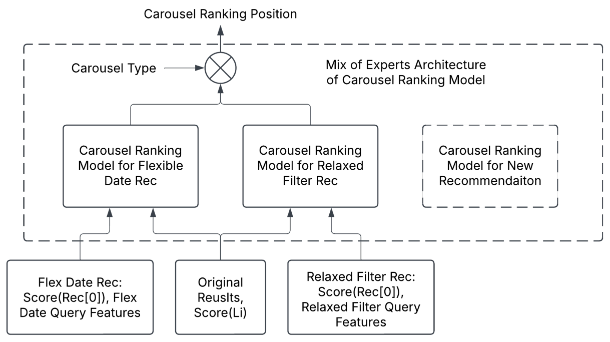
Carousel Ranking Position
Carousel Type
Mix of Experts Architecture of Carousel Ranking Model
Carousel Ranking Model for Flexible Date Rec
Carousel Ranking Model for Relaxed Filter Rec
Carousel Ranking Model for New Recommendaiton
Flex Date Rec: Score(Rec[0]), Flex Date Query Features
Original Reuslts, Score(Li)
Relaxed Filter Rec: Score(Rec[0]), Relaxed Filter Query Features

Figure 3: Architecture of Carousel Ranking Model for Multi-Recommendation.

As the recommendation system is modular and extensible, we can gradually introduce additional types of recommendations. To enhance the sensitivity of experiments, we

- keep the launched carousel ranking model unchanged and only introduce sub-networks for new carousels in the MoE architecture (Fig 3).
- leverage Algorithm 3 to limit experiment exposure to guests who are likely to see the difference introduced by the new carousels.

This prevents over-experiment exposure to guests whose searches do not involve the new types of recommendations, or cases where new carousels are insert to later pages where guest has not requested. The impact of this approach is described in Section 6.2

Table 1: Flexible date carousel ranking experiment: from rule-based to model-based approach on a single recommendation

Experiment	Baseline	Uncancelled Bookers	Carousel Viewers	Listing Detail Page Viewers	Low Inventory Searchers	PR AUC	Precision	Recall
Rule ($\alpha = 0.01$)	Rule ($\alpha = 0.5$)	0.35%	52.18%	0.09%	-0.19%	0.53	0.28	0.79
NN Higher Precision	Rule ($\alpha = 0.01$)	0.21%	-0.39%	-0.05%	-0.035%	0.58	0.33	0.77
NN Higher Recall	Rule ($\alpha = 0.01$)	Not Stat-Sig	1.37%	Not Stat-Sig	Not Stat-Sig	0.58	0.27	0.83

Algorithm 3 Algorithm for A/B Testing New Carousel Types

```
Inputs:
Existing Recommendations $\{\mathbf{Rec}_t, t \in \mathcal{T}\}$
New Type of Recommendation $\mathbf{Rec}_{\mathcal{T}+1}$
Guest Requested Total Pages $\mathcal{P}$
Steps:
Compute $\mathbf{Pos}_{\text{Existing}}$ using Algorithm 2 for $\{\mathbf{Rec}_t, t \in \mathcal{T}\}$.
Compute $\mathbf{Pos}_{\text{w\_New}}$ using Algorithm 2 for $\{\mathbf{Rec}_t, t \in \mathcal{T}+1\}$.
Compare $\mathbf{Pos}_{\text{Existing}}$ and $\mathbf{Pos}_{\text{w\_New}}$ for difference up to page $\mathcal{P}$.
if differences exist then
    Determine if the guest belongs to control or treatment group:
    if guest in treatment group then
        Using $\mathbf{Pos}_{\text{w\_New}}$ to insert $\{\mathbf{Rec}_t, t \in \mathcal{T}+1\}$
    else
        Using $\mathbf{Pos}_{\text{Existing}}$ to insert $\{\mathbf{Rec}_t, t \in \mathcal{T}\}$.
    end if
end if
```

6 Experiment result

Since the launch of flexible date recommendations in September 2023, multiple iterations in augmenting search result with recommendations has lead to a cumulative of 1.2% increase in uncancelled bookings in Airbnb. This section highlights experiments for developing model-based carousel ranking and expanding from single to multi-carousel ranking.

6.1 Single Carousel Ranking Experiment

Table 1 shows iterations in developing a model-based carousel ranking from a rule-based. As described in Section 5.1.5, we started from relaxing the threshold (α from 0.5 to 0.01) in the formula: $\text{Score}(\mathbf{Rec}[0]) - \text{Score}(L_i) > \alpha$. This relaxation significantly increased the visibility of $\mathbf{Rec}_{flex\_date}$, as captured by the metric "Carousel Viewers." It allowed guests to navigate away from situations with insufficient search results (as evidenced by a reduction in "Low Inventory Searchers"), which subsequently led to more "Listing Detail Page Viewers" and, ultimately, an improvement in the primary metric of "Uncancelled Bookers."

To train the model based carousel ranking, we created pairs of listings—one from **Rec** and one from $\mathcal{R}$ in the same search page. A positive label was assigned when the listing from **Rec** led to a booking while the listing from $\mathcal{R}$ did not, and vice versa for negative labels. The training dataset exhibited an imbalanced positive-to-negative sample ratio (about 1:20), so we applied a higher weight to the loss function for positive samples. Cross-entropy was used as the loss function, and PR-AUC, Precision, and Recall were chosen as the offline evaluation metrics.

Table 1 presents two trained models, both achieving an increase in PR-AUC 9% compared to the rule-based approach. A/B testing revealed that the higher-precision model performed better due to its more balanced ranking of **Rec** and $\mathcal{R}$. Although overall "Carousel Viewers" decreased compared to the baseline, the reduction in "Low Inventory Searchers" and the increase in "Uncancelled Bookers" demonstrate that the high-precision model is both more effective at identifying scenarios where flexible date recommendations offer meaningful additional inventory and minimizes distractions for guests in situations where **Rec** does not provide a significant advantage over $\mathcal{R}$.

To further interpret the model's behavior, we used Individual Conditional Expectation (ICE) plots [6], as shown in Fig 4. By sampling search sessions and sweeping a single feature while holding others constant, we visualized model scores under varying conditions. For longer stays (e.g., nights increase from 1 to 28), model considers $\mathbf{Rec}_{flex\_date}$ more valuable. This trend aligns with the observation that longer trips involve higher costs, making guests more price-sensitive and flexible with travel dates. $\mathbf{Rec}_{flex\_date}$ often provide better-priced, higher-quality listings for such cases. However, for stays exceeding 28 nights, the model became more conservative, likely due to sparse training data in the very long trip durations. Additionally, the model prioritizes showing **Rec** for searches with short lead times and low $|\mathcal{R}|$, aligning with our expectations of scenarios where **Rec** offers meaningful values to guests.

6.2 Multi-carousel Ranking Experiment

Table 2 highlights experiments involving the addition of relaxed filter recommendations (e.g., amenities or price). As these recommendations apply to a narrower subset of searches, their total impacts are smaller. Consequently, minimizing dilution effects during A/B testing is crucial to detect incremental changes. By leveraging Algorithm 3 to limit experiment overexposure, we achieve greater sensitivity in detecting changes during experiments introducing relaxed filter recommendations, compared to experiments without narrowly triggered setups. This is evident in the lower values observed in the "Uncancelled Bookers x Coverage" metric—which estimates the impact of "Uncancelled Bookers" across all guests—in experiments involving relaxed amenities and prices.

For relax amenities filter recommendations, we began with a rule-based ranking approach and subsequently trained a ranking model based on collected data. To accelerate experimentation timelines, we bundled relaxed amenities with relaxed price recommendations. The initial rule-based testing on relaxed amenity recommendation validated our hypothesis that without a trained model, the new

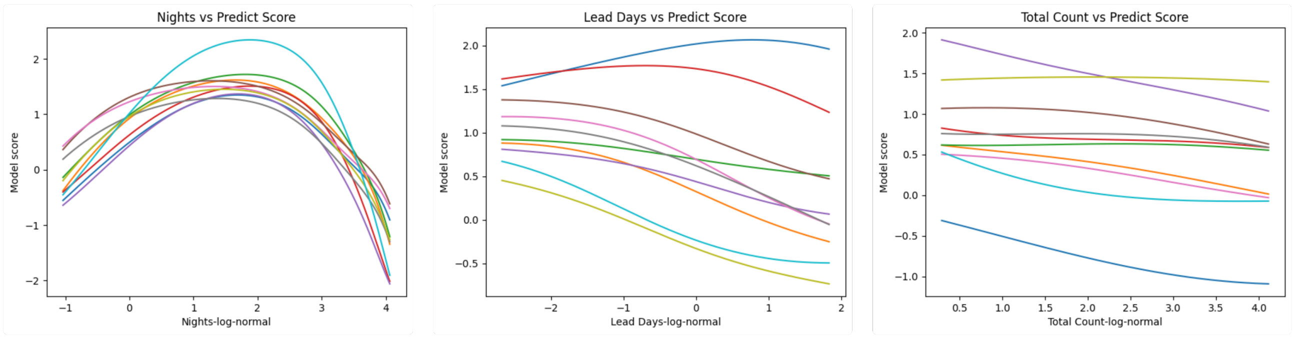

Figure 4: ICE plots illustrating how carousel ranking model scores vary based on features such as Length of Stay, Lead Days, and Total Original Result Count.

Table 2: Multi-type recommendation carousel ranking experiments: extensible for multiple recommendations

Experiment (Carousels and Ranking)	Baseline	Experiment Coverage	Experiment Duration / MDE	Uncancelled Bookers (UB)	UB x Coverage
$\mathbf{Rec}_{flex\_date}$ Rank by Neural Network (NN), not narrowly triggered	$\mathbf{Rec}_{flex\_date}$ Rule	81.2%	5 weeks / 0.24%	0.21%	0.17%
$\mathbf{Rec}_{flex\_date}$ NN + Relax Amenity (Rule, High Pri)	$\mathbf{Rec}_{flex\_date}$ NN	21.1%	10 weeks / 0.33%	Not Stat-Sig	Not Stat-Sig
$\mathbf{Rec}_{flex\_date}$ NN + Relax Amenity (Rule, Low Pri)	$\mathbf{Rec}_{flex\_date}$ NN	19.0%	10 weeks / 0.33%	0.27%	0.051%
$\mathbf{Rec}_{flex\_date}$ NN + Relax Amenity (NN, High Pri) + Relax Price (Rule, Low Pri)	$\mathbf{Rec}_{flex\_date}$ NN	29.7%	4.5 weeks / 0.35%	0.33%	0.098%

carousel was better suited for the low-priority queue in the multi-carousel ranking (Algorithm 2). This can be explained by the fact that, in scenarios where multiple recommendation carousels compete for the most visible positions, the model-based approach is more accurate in determining whether **Rec** is better than $\mathcal{R}$. By placing rule-based carousels C' in the lower-priority queue, they are shown only when models predict that model-ranked carousels C do not help guest bookings. This setup ensures incremental gains without causing cannibalization.

Therefore, in our most recent experiment, we placed flexible date and relax amenities filter recommendations, backed by trained carousel ranking model, in the high-priority queue. Meanwhile, the relaxed price recommendation, using rule-based ranking, was placed in the low-priority queue. This combined approach achieved both greater gains in "Uncancelled bookers" and significantly shortened experimentation periods, reducing the timeline from 10 weeks to 4.5 weeks.

7 Conclusion and Future Work

In summary, we presented a modular and extensible architecture to augment search results with recommendations, assisting guests in finding suitable travel listings by automatically suggesting options aligned with their broader travel intent. The carousel ranking algorithm efficiently leverages the existing search ranking system's listing rankings. Through transfer learning, we enhanced the ability to compare booking probabilities between recommended listings and original results. This approach allows the system to display recommendations only when they provide meaningful value in helping guests find listings that facilitate bookings, while avoiding recommendations that could distract guests.

The extensibility exists both in the architecture and the multi-recommendation carousel ranking algorithm, enabling the efficient introduction and scaling of new recommendation types. This ensures the system continuously evolves with innovative ideas to help guests discover better results. Additionally, the modularized design facilitates further improvements in recommendation query generation, recommendation listing ranking, and carousel ranking. These future developments will be shared in upcoming work.

Acknowledgments

We appreciate the thoughtful discussion with Xin Liu and Malay Haldar. We also acknowledge the invaluable collaboration and support of our cross-functional and partner teammates from various teams: Sherry Chen, Clarence Quah, Sylvia Liu, Aditya Vohra, and Matt Yun.

References

[1] Apache Lucene 2025. *Apache Lucene.* Retrieved May 3, 2025 from https://lucene.apache.org/
[2] Chris J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

[3] Dillon Davis, Huiji Gao, Thomas Legrand, Malay Haldar, Alex Deng, Han Zhao, Liwei He, and Sanjeev Katariya. 2024. Transforming Location Retrieval at Airbnb: A Journey from Heuristics to Reinforcement Learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*. ACM, 4454–4461. doi:10.1145/3627673.3680089

[4] Malay Haldar, Mustafa Abdool, Liwei He, Dillon Davis, Huiji Gao, and Sanjeev Katariya. 2023. Learning To Rank Diversely At Airbnb. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*. ACM, 4609–4615. doi:10.1145/3583780.3614692

[5] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. 2019. Applying Deep Learning to Airbnb Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 1927–1935. doi:10.1145/3292500.3330658

[6] Malay Haldar, Prashant Ramanathan, Tyler Sax, Mustafa Abdool, Lanbo Zhang, Aamir Mansawala, Shulin Yang, Bradley Turnbull, and Junshuo Liao. 2020. Improving Deep Learning for Airbnb Search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 2822–2830. doi:10.1145/3394486.3403333

[7] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3, 1 (1991), 79–87. doi:10.1162/neco.1991.3.1.79

[8] Chun How Tan, Austin Chan, Malay Haldar, Jie Tang, Xin Liu, Mustafa Abdool, Huiji Gao, Liwei He, and Sanjeev Katariya. 2023. Optimizing Airbnb Search Journey with Multi-task Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*. ACM, 4872–4881. doi:10.1145/3580305.3599881

[9] Jie Tang, Huiji Gao, Liwei He, and Sanjeev Katariya. 2024. Multi-objective Learning to Rank by Model Distillation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*. ACM, 5783–5792. doi:10.1145/3637528.3671597

[10] The Path to Purchase: Uncovering how travelers plan and book online 2023. *The Path to Purchase: Uncovering how travelers plan and book online.* Retrieved May 3, 2025 from https://partner.expediagroup.com/en-us/resources/research-insights/path-to-purchase