Beyond Pairwise Learning-To-Rank At Airbnb

MALAY HALDAR, Airbnb, Inc., USA
DAOCHEN ZHA, Airbnb, Inc., USA
HUIJI GAO, Airbnb, Inc., USA
LIWEI HE, Airbnb, Inc., USA
SANJEEV KATARIYA, Airbnb, Inc., USA

There are three fundamental asks from a ranking algorithm: it should *scale* to handle a large number of items, sort items *accurately* by their utility, and impose a *total order* on the items for logical consistency. But here's the catch—no algorithm can achieve all three at the same time. We call this limitation the SAT theorem for ranking algorithms. Given the dilemma, how can we design a practical system that meets user needs? Our current work at Airbnb provides an answer, with a working solution deployed at scale.

We start with pairwise learning-to-rank (LTR) models—the bedrock of search ranking tech stacks today. They scale linearly with the number of items ranked and perform strongly on metrics like NDCG by learning from pairwise comparisons. They are at a sweet spot of performance vs. cost, making them an ideal choice for several industrial applications. However, they have a drawback—by ignoring interactions between items, they compromise on accuracy.

To improve accuracy, we create a "true" pairwise LTR model—one that captures interactions between items during pairwise comparisons. But accuracy comes at the expense of scalability and total order, and we discuss strategies to counter these challenges.

Traveling further along the road to greater accuracy, we take each item in the search result, and compare it against the rest of the items along two dimensions: (1) *Superiority*: How strongly do searchers prefer the given item over the remaining ones? (2) *Similarity*: How similar is the given item to all the other items? This forms the basis of our "all-pairwise" LTR framework, which factors in interactions across all items at once. Looking at items on the search result page all together—superiority and similarity combined—gives us a deeper understanding of what searchers truly want. We quantify the resulting improvements in searcher experience through offline and online experiments at Airbnb.

1 Introduction

Search ranking orders a corpus of items to maximize some combination of utilities for the searcher. It's the tool we rely on daily to cut through endless choices—whether it's finding some information, a meal, a job, a vacation spot, or a place to stay. At the heart of search ranking is the learning-to-rank (LTR) algorithm, which learns from past searcher decisions to improve future results. A straightforward way to learn from past examples is to frame it as a binary classification task, where items that satisfied the searchers' needs are treated as positive, the remaining choices that were presented to the searcher are labeled negative, and a model is trained to distinguish between the two. However a lot of industrial scale applications use, not the simple binary classification, but a variant of the formulation known as pairwise LTR.

Authors' addresses: Malay Haldar, Airbnb, Inc., USA, malay.haldar@airbnb.com; Daochen Zha, Airbnb, Inc., USA, daochen.zha@airbnb.com; Huiji Gao, Airbnb, Inc., USA, huiji.gao@airbnb.com; Liwei He, Airbnb, Inc., USA, liwei.he@airbnb.com; Sanjeev Katariya, Airbnb, Inc., USA, sanjeev.katariya@airbnb.com.

The seminal paper that introduced the Bradley-Terry-Luce model and analyzed pairwise LTR appeared in 1957 [5], with neural network applications to the problem emerging in a 1994 paper [3]. Since then, pairwise LTR has found adoption at major internet companies [23], [24], [4], [25], including Airbnb [8]. What has kept this decades old area of research active and thriving? It's the increasing availability of data, advances in computational power, and the growing business value of improved rankings. This paper presents one step forward in this long narrative arc, highlighting recent advancements in ranking at Airbnb that go beyond traditional pairwise LTR.

We start by examining the properties that make pairwise LTR scalable and effective. The formalism introduced in Section 3 is a stepping stone for the sections ahead, where we address the limitations of pairwise LTR that impact its accuracy. We make improvements in two steps, first considering interactions between two listings at a time in Section 4, then in Section 5 scaling up to interactions across all listings. Section 6 extends the framework from optimizing for a single objective, to optimizing for multiple objectives. We cite related work embedded in context of the discussion throughout the paper. The final section covers experiments performed at Airbnb to compare the frameworks developed, and their outcomes. But first, the SAT theorem.

2 SAT Theorem for Ranking Algorithms

A practical ranking algorithm needs to meet three requirements:

- *Scalability*: From a systems perspective, ranking must be able to handle thousands to millions of items—typically within a latency budget of $O(100)$ milliseconds.
- *Accuracy*: From a user perspective, items with higher utility should rank higher. This ensures the most useful items appear in positions that get the most attention. Accuracy is evaluated using ranking metrics like NDCG and MAP.
- *Total order*: To make ranking mathematically well-defined, it needs to create a total order [30]. This means for any two items, a and b:
  (1) $a \leq a$ (reflexive).
  (2) If $a \leq b$ and $b \leq c$ then $a \leq c$ (transitive).
  (3) If $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetric).
  (4) $a \leq b$ or $b \leq a$ (strongly connected).

In the spirit of the CAP theorem [29] for distributed systems, the SAT theorem states that a ranking algorithm can only fulfill two of these three requirements at a time. Tversky's insights from choice theory [27] point to the root cause for this tension—item utility depends on other items, which is hard to model while guaranteeing total order and scalability.

We start by describing how pairwise LTR deals with the SAT theorem.

3 Pairwise LTR: A Brief Overview

Pairwise LTR is at the core of search ranking at Airbnb. Users search by entering a location, check-in/out dates, and the number of guests. In response, the ranking algorithm organizes the available listings based on several factors. How this ranking algorithm evolved over the years is captured in [8], [9], [26], [7], and [10].

The base objective of the pairwise LTR model is to order the listings by their probability of getting booked. The model is trained on pairs of booked and not-booked listings extracted from logged search results following this setup:

l_{booked} : features of the booked listing
$l_{notBooked}$: features of the listing not booked
$f(l)$: an MLP (multi-layer perceptron) with listing l as input.
$\theta(x) = (1 + e^{-x})^{-1}$, the sigmoid function

The training process minimizes the loss:

$$loss = -log(\theta(f(l_{booked}) - f(l_{notBooked}))) \tag{1}$$

which is the binary cross-entropy loss with label set to 1. Throughout the paper, MLPs like $f(\cdot)$ also take user and query features as input, but we've skipped them for readability. After training, the model produces two quantities:

$$\text{Pairwise logit, } logit(a, b) = f(l_a) - f(l_b) \tag{2}$$

$$\text{Pairwise probability, } P(l_a > l_b) = \theta(logit(a, b)) \tag{3}$$

For ranking N listings, pairwise LTR assigns each listing l_i a score using $f(l_i)$ and then sorts listings by their scores. Since $f(\cdot)$ is *univariate*, scoring one listing at a time, pairwise LTR scales linearly. The MLP $f(\cdot)$ provides a deterministic mapping from listing features to a real valued score. The total order of listings can be obtained from the total order of the corresponding scores.

The place where pairwise LTR misses out is accuracy. Here's why. Ordering the listings by $f(\cdot)$ is equivalent to ordering them by the pairwise booking probabilities. The Bradley-Terry equation links these pairwise booking probabilities to the individual (pointwise) booking probabilities $P(l_a)$ and $P(l_b)$:

$$P(l_a > l_b) = P(l_a)/(P(l_a) + P(l_b)) \tag{4}$$

Hence, $f(l_a) > f(l_b)$ implies $P(l_a) > P(l_b)$, and pairwise LTR ends up ordering listings by their pointwise booking probabilities. The key limitation here is that these probabilities are treated as if they depend only on the listing itself. In reality, the booking probability of a listing is influenced by the entire set of N listings shown to the user [27].

For a more precise understanding of pairwise LTR's limitations, let's focus on the properties of the pairwise logit described in Eq 2. If we take three listings, l_a, l_b, and l_c, it's easy to show that:

$$logit(a, b) + logit(b, c) = logit(a, c) \tag{5}$$

This proves that for a set of N listings, all the $N(N-1)/2$ pairwise logits align along a single straight line. This characteristic is known as collinearity. Collinearity *does not* mean that pairwise LTR is learning a linear relationship between the input features of the listings.
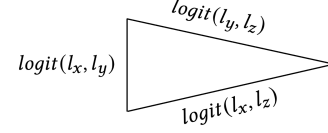

Fig. 1. The relation $logit(l_x, l_y) \gg 0 \wedge logit(l_x, l_z) \approx logit(l_y, l_z)$ cannot be captured in a 1D space. It is represented by the triangle inequality $logit(l_x, l_y) + logit(l_y, l_z) > logit(l_x, l_z)$.

Those relationships are actually managed by $f(\cdot)$, a non-linear MLP. Collinearity simply indicates that the pairwise-logit space—created by comparing all listing pairs—is limited to one dimension.

This restriction on the logit space prevents pairwise LTR from accurately modeling booking probabilities for users. Let's break it down with an example. Imagine two listings, l_x and l_y, that are identical apartments in the same building, but l_x is $10 cheaper. Naturally, most searchers would prefer l_x, so $logit(l_x, l_y)$ would have a high positive value, i.e., $logit(l_x, l_y) \gg 0$.

Now, add a third listing, l_z, that's quite different in terms of location, price, and amenities. The values of $logit(l_x, l_z)$ and $logit(l_y, l_z)$ would probably be similar because users who favor l_x over l_z would also favor l_y over l_z. This means $logit(l_x, l_z) \approx logit(l_y, l_z)$. If we apply collinearity, we'd get

$$logit(l_x, l_y) + logit(l_y, l_z) \approx logit(l_y, l_z)$$

which implies $logit(l_x, l_y) \approx 0$. This contradicts the earlier result of $logit(l_x, l_y) \gg 0$, proving that collinearity doesn't apply in this case. Figure 1 illustrates why pairwise comparisons can't always fit into a 1D logit space.

4 True-Pairwise LTR

In order to pursue accuracy, we break free from the restrictive 1D logit space of pairwise LTR and build a "true" pairwise model $g(l_a, l_b)$. Unlike the *univariate* pairwise LTR model $f(\cdot)$, the true-pairwise model $g(\cdot)$ is *bivariate*. It takes both the listings being compared as inputs, allowing it to learn interactions between the two listings. When training this model on an example pair of booked and not-booked listings $\{l_{bk}, l_{nbk}\}$, the loss is computed as:

$$loss = -log(\theta(g(l_{bk}, l_{nbk}))) - log(1 - \theta(g(l_{nbk}, l_{bk}))) \tag{6}$$

which is the symmetric binary cross entropy loss for predicting that l_{bk} is preferred over l_{nbk}.

The pairwise logit is directly implemented by the true-pairwise LTR model as:

$$logit(l_a, l_b) = g(l_a, l_b) \tag{7}$$

Because $g(\cdot)$ is a general bivariate function, it can represent the types of relationships shown in Figure 1. As an MLP, $g(\cdot)$ is flexible enough to learn any interaction between l_a and l_b. This improves the accuracy of the pairwise booking probability $\theta(g(l_a, l_b))$. However, as stated by the SAT theorem, this increase in accuracy comes at a cost: reduced scalability and the loss of total order. We dive into strategies to restore total order and mitigate the loss of scalability next.

4.1 Total Order

The true-pairwise LTR model doesn't naturally enforce the conditions needed for total order (described in Section 2). To tackle this, we make $g(\cdot)$ anti-commutative, meaning: $g(l_a, l_b) = -g(l_b, l_a)$. Enforcing anti-commutativity is a practical way to guarantee three of the four conditions—reflexivity, antisymmetry, and strong connectivity—as directly enforcing them is hard. To make $g(\cdot)$ anti-commutative, we define it as:

$$g(l_a, l_b) = h(l_a, l_b) - h(l_b, l_a) \tag{8}$$

Here, $h(\cdot)$ is an MLP that learns interactions between the two listings. This approach ensures $g(l_a, l_b) = -g(l_b, l_a)$ while keeping the flexibility to capture complex relationships between l_a and l_b. Figure 2a illustrates how the true-pairwise LTR model is trained.

Even with anti-commutativity in place, we still need to enforce transitivity to fully establish total order. Without transitivity, $g(l_a, l_b)$ cannot be directly plugged into popular sorting algorithms like Quicksort or Mergesort.

To solve this problem given N listings, for each listing l_i, we collect the $(N-1)$ scores $g(l_i, l_1), g(l_i, l_2), \ldots, g(l_i, l_n)$. Given that we have established anti-commutativity for $g(\cdot)$, we can apply a simpler version of the generalized Bradley-Terry model (Eq 10 from [12]):

$$score(l_i) = \frac{1}{1 + \sum_{j \neq i, j=1}^{N} e^{-g(l_i, l_j)}} \tag{9}$$

These scores assign each listing a real value and are invariant to how the listings are ordered. Since real-valued scores naturally fulfill the requirements for total order, they can now be used with standard sorting algorithms like Quicksort or Mergesort. Figure 2b illustrates how a listing is scored in the true-pairwise LTR framework.

Our approach can be contrasted with the true-pairwise LTR method proposed in [17], where the $(N-1)$ scores are averaged for a listing. By leveraging the generalized Bradley-Terry model, we achieve a higher accuracy of booking probability due to the performance bounds provided in [12]. The two approaches are compared under experimental results.

4.2 Scalability

With N listings, true-pairwise LTR requires $N * (N-1)/2$ unique evaluations of $g(\cdot)$, one for each pair of listings. In contrast, pairwise LTR only needs N evaluations. Because of its quadratic complexity, we use true-pairwise LTR only in the final stage of a multi-stage ranking system, after the number of listings has been reduced to $O(10)$. This leads to a hybrid system where the first-stage ranker is still owned by pairwise LTR. Figure 3 outlines the complexity levels across different ranking stages. For more on algorithms suited to the final ranking stage, see [16].

5 All-Pairwise LTR

Bivariate true-pairwise LTR improves the accuracy of univariate pairwise LTR by accounting for interactions between two listings being compared. Building on this idea, we introduce a *multivariate* LTR framework that considers how each listing interacts with all other listings at once. This approach better aligns with user behavior described in choice theory [27] and leads to improved accuracy of booking probability predictions.

Scoring a listing in true-pairwise LTR (Figure 2b) consists of two steps:

- Calculating the $(N-1)$ pairwise logits using the true-pairwise LTR model $g(\cdot)$.
- Combining these pairwise logits into one final score using the generalized Bradley-Terry model (Eq 9).

In all-pairwise LTR, we replace both these steps by a single model. This model not only takes the listing being scored as input, but the remaining $(N-1)$ listings as well to compute interactions.

The generalized Bradley-Terry model, used in the second step of true-pairwise LTR, lacks adaptability since it doesn't have learnable parameters. To address this, we replace it with an MLP, named the all-pairwise logit network (APLN). During training, APLN generates logits, which we pair for booked and non-booked listings, and then optimize the loss in Eq 1.

Next we turn our attention to the features that should feed the APLN network. To efficiently capture how a listing interacts with the other $(N-1)$ listings, we create two types of features:

- *Superiority features*: Due to its $O(N^2)$ complexity, the all-pairwise LTR model serves as the second-stage ranker, re-ordering the top N. We still have the pairwise LTR model as the first-stage ranker, and therefore the pairwise LTR logits available for each listing. We reuse these logits from the first-stage ranker as features for the second-stage ranker. We take pairwise differences of the logits and apply a sigmoid, generating a feature in the $(0, 1)$ range. This in effect captures the relative superiority of the current listing compared to the rest.
- *Similarity features*: The superiority features fail to capture a crucial piece of information—the similarity between listings. Note the violation of collinearity discussed in Section 4 relied on similar listings. Accounting for listing similarities is essential to diversify search results, discussed in our previous work [7]. To create similarity features, we first map the listing feature vectors to embeddings. Dot product of the embeddings measure similarity, and a softmax normalizes the dot product to the $(0, 1)$ range. In our earlier work [7], we manually built a scoring function on top of these similarities. In all-pairwise LTR, we let the APLN learn this scoring function.

But before we can feed the superiority-similarity features into the APLN, we need to overcome one more obstacle. Recall that to implement a valid comparator, the true-pairwise LTR model needed to be anti-commutative. For all-pairwise LTR where we input multiple listings, this requirement generalizes to ensuring input permutation invariance.

If superiority-similarity features are directly fed into the APLN network, input permutation invariance cannot be guaranteed. To see this, consider a single neuron of the input layer of APLN. Let the ith input (e.g. superiority over l_i) be x_i, the ith weight be w_i, and b be the layer bias. The neuron's output, $x_i * w_i + b$, changes if l_i is reordered. To make the neuron's output invariant to input order, we make the ith weight w_i depend on l_i. This dependency is modeled through a function $\phi(l_i)$, implemented as an MLP. The
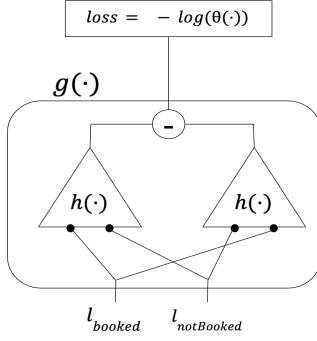
(a) True-pairwise training, optimizing the loss from Eq 6.
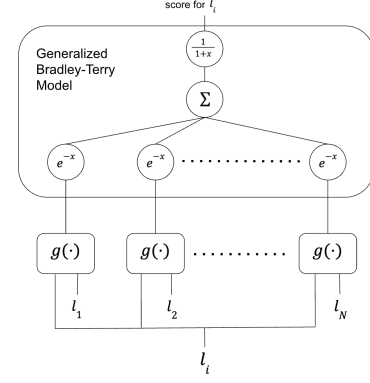
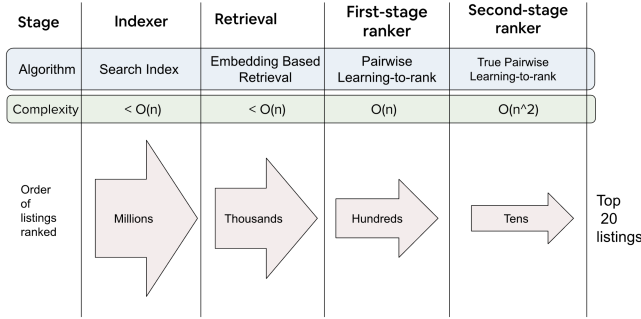(b) True-pairwise scoring utilizing Eq 9.

Fig. 2. True-pairwise LTR

Fig. 3. Anatomy of a multi-stage ranker.

modified neuron output, written as $x_i * \phi(l_i) + b$, is now permutation invariant.

An alternative way to achieve permutation invariance, proposed in [1], involves generating and aggregating permutations of item subsets. In our approach, we avoid the computational complexity of creating permutations. The superiority-similarity features, along with the layer imposing input permutation invariance, make up the all pairwise feature network (APFN). Construction of superiority features is outlined in Figure 4a, while Figure 4b illustrates the similarity features. The overall all-pairwise LTR architecture is shown in Figure 5.

We designed APFN to learn how the scored listing interacts with the other $N-1$ listings. An alternative to APFN is to use an attention network to construct the features, as explored in [19], [28] and [18]. This gives us an off-the-shelf solution which is easier to get started with. In the experimental results section to follow, we compare how these methods stack up against each other.

Interactions between the listings scored can also be modeled using RNNs, as shown in [31] and [2]. Such methods, however, are not input permutation invariant and cannot guarantee total order. Their learning is dependent on the input sequences seen during training and they don't learn a generalized sorting capability.

Another interesting point of comparison for all-pairwise LTR is with neural networks that implement differentiable sorting functions, such as [6] and [20]. These too are $O(N^2)$ complexity solutions capable of ranking listings, but their focus is to replace traditional sorting algorithms like Quicksort with a neural network. In contrast, the goal of all-pairwise LTR is to improve ranking by learning interactions between listings, while still using Quicksort or Mergesort to perform the final sorting.

A frequent stumbling block is the apparent resemblance between all-pairwise LTR and its distant cousin, listwise LTR [4], a univariate model that optimizes the softmax cross-entropy loss. Since listwise LTR is univariate, it cannot account for interactions between listings. Similarly, methods that directly optimize NDCG [21], [13] also rely on univariate models, leaving them unable to model listing-to-listing interactions.

All-pairwise LTR maps each listing to a real valued score in a way that doesn't depend on the order of inputs; these scores subsequently provide a total order. To address scalability, we limit all-pairwise LTR to the second-stage ranking and use it alongside a pairwise LTR model for the first stage. While none of the individual models in this hybrid system can escape the SAT theorem, the system as a whole reaches a high degree of accuracy while maintaining total order and scalability.

6 Optimizing Multiple Objectives

So far, we've focused on ranking the listings by their booking probability. For ranking systems serving internet-scale applications, conversion is typically the principal, but still one of many objectives. Airbnb is no exception and ranking aims to minimize a range of negative outcomes, while trying to maximize positive outcomes for a trip. As an example to explore multiple objectives, we use trip rating—the 5-star reviews guests leave after their stay.

Determining the relative weights for objectives like booking and trip rating is an extensive topic, and beyond the scope of this paper. For our purposes, we will assume that each objective is given a weight, and the goal of ranking is to maximize the weighted sum of the objectives.
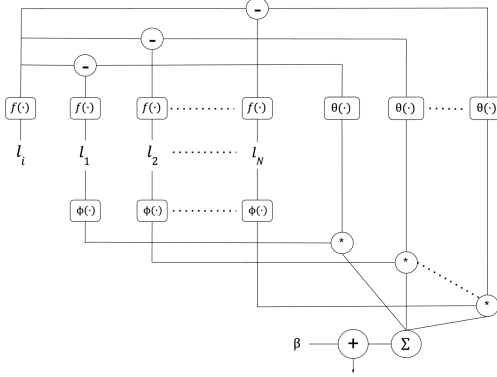
(a) Feature computing the superiority of the given listing l_i over the remainder of listings $l_1, l_2, \ldots l_N$. $f(\cdot)$ is the pairwise LTR first-stage ranker represented by Eq 2. $\theta(\cdot)$ is the sigmoid function. $\phi(\cdot)$ is an MLP which takes the listing features as input. β is the bias term.
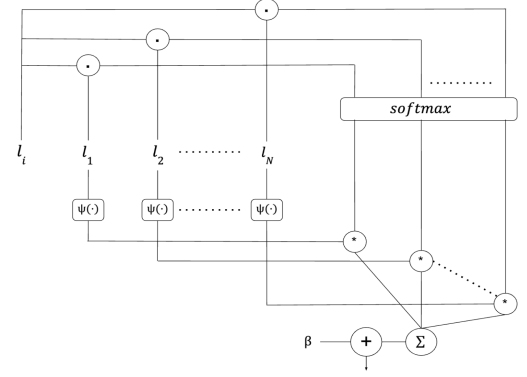
(b) Feature computing similarity of the given listing l_i to the remainder of listings $l_1, l_2, \ldots l_N$ by taking dot products. $\psi(\cdot)$ is an MLP which takes the listing features as input. β is the bias term.

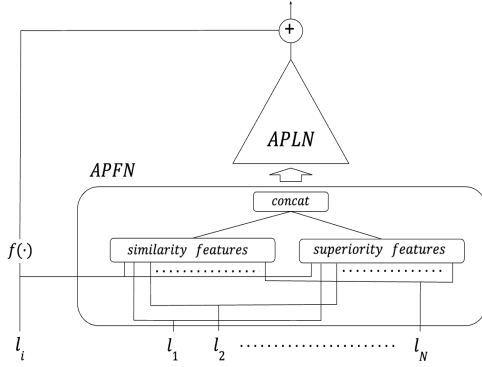Fig. 4. Superiority and similarity features of APFN

Fig. 5. The all-pairwise architecture. *Superiority features* refers to Figure 4a. *Similarity features* refers to Figure 4b. Together they form the all pairwise feature network (APFN). The APFN feeds into the all pairwise logit network (APLN), an MLP. To produce the final logit, APLN acts as a residual to the logit produced by the pairwise LTR first-stage ranker $f(\cdot)$.

A popular solution to the problem is multi-task learning ([22], [26]), where a model predicts multiple objectives through different heads and combines them using weights. This approach makes the model more complex, since it has to handle multiple labels and balance dense and sparse ones. Particularly in our case, true-pairwise and all-pairwise LTR fit only one of the objectives, the booking probability, as searchers naturally compare various listings in search results. These methods aren't suitable for predicting trip ratings, requiring a separate model to handle that task.

Using separate models for each objective creates challenges—like conflicting outputs and missing interactions between objectives.

We discard the multi-task and multi-model approaches to solve the multiple objectives puzzle. Instead, we unify the objectives by adjusting the training loss. For each training example consisting of a pair of listings l_{booked} and $l_{notBooked}$, we compute a weight:

$$\omega_{total} = 1 + \omega_{tripQuality}(l_{booked}) \qquad (10)$$

where a booking get a base weight of 1, and trip quality adds to it depending on the reviews of l_{booked}. The loss is calculated as:

$$loss = -log(\theta(f(l_{booked}) - f(l_{notBooked}))) * \omega_{total} \qquad (11)$$

Intuitively, instead of treating all pairwise wins equally, we value each win by its trip quality. This gives us a simple and effective way to target multiple objectives within true-pairwise and all-pairwise LTR. We demonstrate the effectiveness of this approach in the experimental results section.

7 Stability Of Results

Optimizing ranking for an internet-scale product isn't just about improving metrics like bookings or trip ratings; it also involves addressing qualitative factors. One important consideration is ensuring the stability of search results. In all-pairwise LTR, a listing's fate is intricately tied to its companions in the lineup; any change in the input set can cause the listing scores to change. The input set of listings may change because of innocuous reasons—like small map movements or simple query date refinements. If such minor changes cause noticeable shifts in search results, it creates a jarring user experience.

To enhance the stability of search results, we leverage the pairwise LTR logits generated by the first-stage ranker. Since this ranker is a univariate function that evaluates one listing at a time, its logits are inherently stable. For the second-stage ranker, we treat the all-pairwise LTR model as a residual adjustment layered on top of the first-stage pairwise LTR model (refer to Figure 5). By making the multivariate logit an incremental adjustment to the stable univariate logit, we effectively stabilize the overall search results. The experimental results section highlights the improvements in stability achieved using this approach.

8 Experimental Results

For our experimental results, we treat pairwise LTR as the baseline and compare it against true-pairwise and all-pairwise LTR. For true-pairwise LTR, we evaluate two versions:
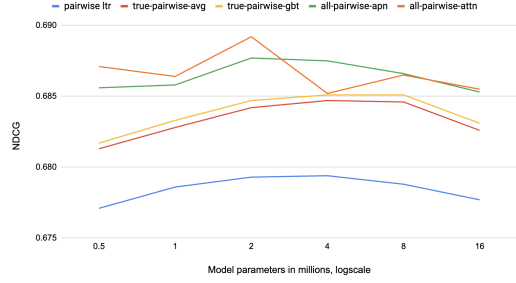
Fig. 6. X-axis: Number of model parameters. Y-axis: Model NDCG.

- *true-pairwise-avg*: The final score for a listing is the average of the $(N - 1)$ scores from true pairwise comparisons.
- *true-pairwise-gbt*: The $(N - 1)$ true pairwise comparisons are aggregated using the generalized Bradley-Terry model (Eq 9).

For all-pairwise LTR, we evaluate two versions as well:

- *all-pairwise-apfn*: This uses the architecture in Figure 5, where the all-pairwise logit network is fed with superiority-similarity features from APFN.
- *all-pairwise-attn*: This substitutes the APFN with an attention network.

All the models are trained on the same training data, which includes pairs of booked and not-booked listings extracted from search logs spanning a year.

We start with offline experiments, using NDCG as the evaluation metric. Booked listings are assigned a relevance of 1, while not-booked listings are assigned a relevance of 0. Given the success of scaling laws for neural language models ([14], [11]), a frequently asked question is whether such scaling is applicable to our ranking models as well. Instead of investigating different LTR frameworks, can ranking performance be improved by simply scaling the model parameters of the baseline pairwise LTR model? In our first offline experiment, we study the effect of scaling the number of trainable model parameters (Figure 6).

The results show that performance improvements obtained by LTR frameworks that better represent searcher behavior, cannot be achieved by simple scaling laws. All-pairwise LTR outperforms true-pairwise LTR, which in turn performs better than pairwise LTR, confirming the increase in accuracy predicted by theory. This creates a clear hierarchy: the more effectively a model captures interactions between listings, the better its ranking performance. Additionally, scaling beyond 2 to 4 million parameters causes overfitting across all variants, showing that indiscriminate scaling can harm performance.

Our second offline experiment focuses on scalability, where we examine the trade-off between cost and benefit for all-pairwise LTR. Figure 7 plots the relationship between the number of listings reranked by all-pairwise LTR in the second-stage ranker, and the resulting change in NDCG and latency. The growth in NDCG is sublinear and flattens beyond a certain point, but the growth in latency continues to rise dramatically. Consequently, there is an optimal trade-off between benefit vs cost, and we settled at reranking 60 listings in the second stage ranker.
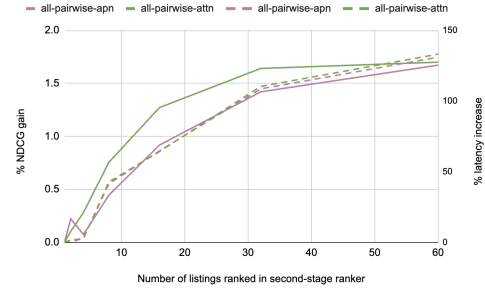

Fig. 7. X-axis: Number of listings reranked. Y-axis left: % change in NDCG with reranking one listing as baseline. Y-axis right: % change in latency.
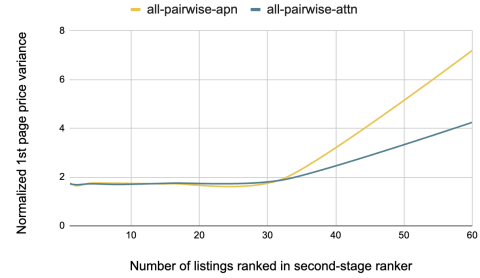

Fig. 8. X-axis: Number of listings reranked. Y-axis: Normalized variance of listing prices on first page of search results.

Section 5 explains how all-pairwise LTR improves ranking by computing listing similarities and optimizing for diversity. In our next experiment, we examine the impact of all-pairwise LTR on diversity, using the variance in listing prices on the first page of search results as a surrogate metric. Figure 8 shows that diversity goes up as the all-pairwise LTR model is allowed to rerank more of the top listings.

While Figure 6 clearly shows the superior performance of all-pairwise and true-pairwise LTR over pairwise LTR, in our fourth experiment we study a flip side—the uncertainty in model performance. Of the two varieties of uncertainty [15]—aleatoric and epistemic—we focus on measuring epistemic uncertainty, since aleatoric uncertainty is independent of the LTR frameworks. Table 1 shows the range of NDCG obtained for the different LTR frameworks by varying the random seed that controls training data shuffling and neural network parameter initialization. The standard deviation in NDCB is around $2x$ to $6x$ higher compared to pairwise LTR. One possible reason for the higher epistemic uncertainty is that multivariate and bivariate models, with more inputs, can fit a broader range of parameters. However, averaging pairwise comparisons, as implemented in *true-pairwise-avg*, helps lower epistemic uncertainty.

To measure how well offline performance generalizes to user impact, we perform online AB tests with pairwise LTR as control, against true-pairwise and all-pairwise LTR as treatments. We measure the percentage gain in uncanceled bookings over the control, which is shown in Figure 9. Given the statistical nature of the test, the 95% confidence intervals are also shown.

The results in Figure 9 show that offline NDCG does not generalize to online performance uniformly. By itself, NDCG provides an

seed	I	II	III	IV	V
1	0.6776	0.6847	0.6830	0.6859	0.6879
2	0.6777	0.6838	0.6833	0.6856	0.6874
3	0.6776	0.6831	0.6828	0.6872	0.6872
4	0.6772	0.6833	0.6827	0.6867	0.6852
5	0.6773	0.6832	0.6826	0.6875	0.6863
6	0.6779	0.6841	0.6826	0.6864	0.6877
7	0.6776	0.6838	0.6833	0.6867	0.6824
8	0.6781	0.6836	0.6831	0.6865	0.6862
stddev	2.92E−4	5.29E−4	2.92E−4	6.23E−4	18.11E−4

Table 1. NDCG corresponding to variations of the models generated by different random seeds that control the training data shuffling and model parameter initialization. (I) pairwise LTR (II) true-pairwise-avg (III) true-pairwise-gbt (IV) all-pairwise-apfn (V) all-pairwise-attn. The final row computes the standard deviation of NDCG for the corresponding models.
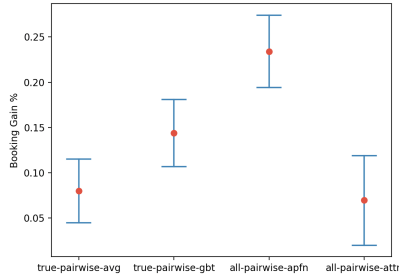

Fig. 9. X-axis: The treatment models. Y-axis: % gain in uncancelled bookings.

incomplete picture of model performance, so the final verdict must rely on key business metrics from online AB tests.

Our next experiment focuses on the multi-objective optimization discussed in Section 6. We pick *all-pairwise-apn*, the most effective ranking model identified so far, and train it with the weighted loss from Eq 11. We compare it with Airbnb's production system, which used separate models to optimize bookings and trip quality. Results of the online AB test are highlighted in Table 2, which demonstrate the effectiveness of training all-pairwise LTR with the loss weighted by multi-objective.

Metric	Percentage Improvement	p-val
Uncanceled Bookings	0.61%	0.00
Uncanceled Nights Booked	0.80%	0.00
Uncanceled Booking Value	1.20%	0.00
Trip Rating Per Booking	0.03%	0.00
Bookings New Guest	0.64%	0.03
Weighted Value	1.30%	0.00

Table 2. Comparison of *all-pairwise-apn* trained with loss of Eq 11 vs active production ranker.

In our final experiment, we measure the improvement in stability discussed in Section 7. To quantify stability, we introduce a small jitter to the map bounds associated with a search query. This introduces small randomized changes to the overall set of listings being

ranked. We then count flips, where a top-ranked listing is replaced by one that was previously outside the top results. Comparing an all-pairwise LTR model without the residual architecture, unstable flips are reduced by −75% with the residual.

	Scalability	Accuracy	Total order
Pairwise	$O(n)$	★	✓
True-pairwise	$O(n^2)$	★★	✗
True-pairwise**	$O(n^2)$	★★	✓
All-pairwise	$O(n^2)$	★★★	✓

Table 3. Comparing LTR frameworks under the SAT theorem. True-pairwise** is modified by Eq 8, followed by generalized Bradley-Terry model.

9 Conclusion

"You can have two Big Things, but not three."

— The Internet.

Of course, the internet is onto something here. But we found a working compromise—combine two things from two places. In our research, all-pairwise LTR as the second-stage ranker, combined with pairwise LTR in the first stage, emerged as the clear winner. Table 3 provides a summary. The all-pairwise LTR model from Section 5, which uses superiority-similarity features and trained with the multi-objective approach from Section 6, was successfully deployed to 100% of Airbnb searchers in early 2025. Prior to the rollout, we conducted an in-depth review of key business metrics, system performance, and its impact on developer productivity. Two questions were identified that remain unanswered: (1) Why all NDCG gains do not generalize equally to booking gains online, and what makes some flavors of NDCG gain more impactful than others? (2) Are there techniques that allow all-pairwise LTR to scale further to handle a larger number of listings? Our current research is focused on these questions.

References

[1] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2019. Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval* (Santa Clara, CA, USA) *(ICTIR '19)*. Association for Computing Machinery, New York, NY, USA, 85–92. https://doi.org/10.1145/3341981.3344218

[2] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2019. *Seq2Slate: Re-ranking and Slate Optimization with RNNs*. Technical Report. https://arxiv.org/abs/1810.02019

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. In *Proceedings of the 7th International Conference on Neural Information Processing Systems* (Denver, Colorado) *(NIPS'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.

[4] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, USA) *(ICML '07)*. 129–136.

[5] L. R. Ford. 1957. Solution of a Ranking Problem from Binary Comparisons. *The American Mathematical Monthly* 64, 8 (1957), 28–33. http://www.jstor.org/stable/2308513

[6] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic Optimization of Sorting Networks via Continuous Relaxations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1eSS3CcKX

[7] 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* (Birmingham, United Kingdom,) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 4609–4615. https://doi.org/10.1145/3583780.3614692

[8] 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* (Anchorage, AK, USA) *(KDD '19)*. 1927–1935.

[9] 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)*. 2822–2830.

[10] Malay Haldar, Hongwei Zhang, Kedar Bellare, Sherry Chen, Soumyadip Banerjee, Xiaotang Wang, Mustafa Abdool, Huiji Gao, Pavan Tapadia, Liwei He, and Sanjeev Katariya. 2024. Learning to Rank for Maps at Airbnb. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 5061–5069. https://doi.org/10.1145/3637528.3671648

[11] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[12] Tzu-Kuo Huang, Ruby C Weng, Chih-Jen Lin, and Greg Ridgeway. 2006. Generalized Bradley-Terry Models and Multi-Class Probability Estimates. *Journal of Machine Learning Research* 7, 1 (2006).

[13] Rolf Jagerman, Zhen Qin, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2022. On optimizing top-k metrics for neural ranking models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2303–2307.

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[15] Alex Kendall and Yarin Gal. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf

[16] Weiwen Liu, Yunjia Xi, Jiarui Qin, Fei Sun, Bo Chen, Weinan Zhang, Rui Zhang, and Ruiming Tang. 2022. Neural Re-ranking in Multi-stage Recommender Systems: A Review. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 5512–5520. https://doi.org/10.24963/ijcai.2022/771 Survey Track.

[17] Franco Maria Nardini, Roberto Trani, and Rossano Venturini. [n. d.]. Learning bivariate scoring functions for ranking. *Discover Computing* 27, 1 ([n. d.]).

[18] Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2020).

[19] Rama Kumar Pasumarthi, Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2020. Permutation Equivariant Document Interaction Network for Neural Learning to Rank. In *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval* (Virtual Event, Norway) *(ICTIR '20)*. Association for Computing Machinery, New York, NY, USA, 145–148. https://doi.org/10.1145/3409256.3409819

[20] Sebastian Prillo and Julian Martin Eisenschlos. 2020. SoftSort: a continuous relaxation for the argsort operator. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 722, 10 pages.

[21] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval* 13 (2010), 375–397.

[22] Yi Ren, Xiao Han, Xu Zhao, Shenzheng Zhang, and Yan Zhang. 2023. Slate-Aware Ranking for Recommendation. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining* (Singapore, Singapore) *(WSDM '23)*. Association for Computing Machinery, New York, NY, USA, 499–507. https://doi.org/10.1145/3539597.3570380

[23] D Sculley et al. 2009. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*, Vol. 3.

[24] Nithish Kannen Senthilkumar, Yao Ma, Gerrit van den Burg, and Jean Baptiste Faddoul. 2024. Efficient pointwise-pairwise learning-to-rank for news recommendation. (2024). https://www.amazon.science/publications/efficient-pointwise-pairwise-learning-to-rank-for-news-recommendation

[25] Amit Sharma and Baoshi Yan. 2013. Pairwise learning in recommendation: experiments with community recommendation on linkedin. In *Proceedings of the 7th ACM Conference on Recommender Systems* (Hong Kong, China) *(RecSys '13)*. Association for Computing Machinery, New York, NY, USA, 193–200. https://doi.org/10.1145/2507157.2507175

[26] 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* (Long Beach, CA, USA,) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 4872–4881. https://doi.org/10.1145/3580305.3599881

[27] Amos Tversky. 1972. Elimination by aspects: A theory of choice. *Psychological review* 79, 4 (1972), 281.

[28] Jianxiong Wei, Anxiang Zeng, Yueqiu Wu, Peng Guo, Qingsong Hua, and Qingpeng Cai. 2020. Generator and Critic: A Deep Reinforcement Learning Approach for Slate Re-ranking in E-commerce. *CoRR* abs/2005.12206 (2020). arXiv:2005.12206 https://arxiv.org/abs/2005.12206

[29] Wikipedia. 2025. CAP theorem. https://en.wikipedia.org/wiki/CAP_theorem [Online; accessed 24-April-2025].

[30] Wikipedia. 2025. Total order. https://en.wikipedia.org/wiki/Total_order [Online; accessed 24-April-2025].

[31] Tao Zhuang, Wenwu Ou, and Zhirong Wang. 2018. Globally optimized mutual influence aware ranking in e-commerce search. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden) *(IJCAI'18)*. AAAI Press, 3725–3731.