# Optimizing Airbnb Search Journey with Multi-task Learning

Chun How Tan
Airbnb Inc.
USA
chunhow.tan@airbnb.com

Austin Chan
Airbnb Inc.
USA
austin.chan@airbnb.com

Malay Haldar
Airbnb Inc.
USA
malay.haldar@airbnb.com

Jie Tang
Airbnb Inc.
USA
jie.tang@airbnb.com

Xin Liu
Airbnb Inc.
USA
xin.liu@airbnb.com

Mustafa Abdool
Airbnb Inc.
USA
moose.abdool@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

## ABSTRACT

At Airbnb, an online marketplace for stays and experiences, guests often spend weeks exploring and comparing multiple items before making a final reservation request. Each reservation request may then potentially be rejected or cancelled by the host prior to check-in. The long and exploratory nature of the search journey, as well as the need to balance both guest and host preferences, present unique challenges for Airbnb search ranking. In this paper, we present Journey Ranker, a new multi-task deep learning model architecture that addresses these challenges. Journey Ranker leverages intermediate guest actions as milestones, both positive and negative, to better progress the guest towards a successful booking. It also uses contextual information such as guest state and search query to balance guest and host preferences. Its modular and extensible design, consisting of four modules with clear separation of concerns, allows for easy application to use cases beyond the Airbnb search ranking context. We conducted offline and online testing of the Journey Ranker and successfully deployed it in production to four different Airbnb products with significant business metrics improvements.

## CCS CONCEPTS

• **Computing methodologies → Multi-task learning**; **Neural networks**; • **Information systems → Learning to rank**; **Content ranking**; *Personalization*; • **Applied computing → Online shopping**.

## KEYWORDS

Search Ranking, Recommender Systems, User Search Journey, Multi-task learning, Two-sided marketplace

**ACM Reference Format:**
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 '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3580305.3599881

## 1 INTRODUCTION

At Airbnb, search is the main way for a guest to discover the right inventory, such as stays, (in-real-life) experiences (i.e. activities to do), online (i.e. virtual) experiences, etc. Specifically, given query parameters from the guest (e.g. location, date range, etc), the Airbnb Search Ranking model ranks all eligible inventories to optimize for the desired business metric. While our proposed model architecture in this paper is applicable to other domains (e.g. four different product use cases within Airbnb), we will focus the majority of our discussion in this paper on one particular instantiation of the model architecture, for the Airbnb's stays product use case.

Unlike other products such as social media feeds, which focus on engagement, Airbnb focuses more on the final guest conversion, similar to other e-commerce and two-sided marketplaces. Furthermore, since booking a place to stay is generally a large expense, guests tend to consider a lot of options carefully, akin to purchasing a big-ticket item such as an electronic appliance, car, or house. Based on internal data, a guest could spend multiple weeks, across dozens of searches, before narrowing down options and making a final reservation request on Airbnb. Lastly, as a two-sided marketplace, the guest's reservation request may end up being rejected or cancelled by the host, resulting in a poor guest experience where the guest must start the whole search journey again.

Figure 1 shows an example guest search journey from a search to a realized reservation check-in. Note that the guest made reservation requests for two listings but ended up being cancelled or rejected by the host, highlighting the importance of considering host preferences. Furthermore, even though the guest clicked on *item-b* early in the search journey, the guest only made a reservation request for *item-b* after more searches, illustrating the long nature of search journeys in the Airbnb product use case.

Thus, Airbnb Stays Ranking needs to guide the guest throughout the long search journey while also balancing both guest and host preferences so that all the stakeholders are satisfied with the final Airbnb stay. However, the exploratory and assorted guest activities at Airbnb present unique challenges in modeling the end-to-end guest search journey. Guests go through different stages, making the modeling problem no longer a single task with binary positive / negative signals. Instead, we need to capture sparse guests and hosts preferences at multiple stages (i.e. milestones) of the search journey, while still taking into account the commonality of guest preference throughout all stages of the search journey.

Furthermore, the mixture of positive (e.g. booking) and negative (e.g. cancellation) stages poses additional challenges. Specifically, the negative stages can be initiated from either guest or host, with varying correlations to other milestones in the same search journey. Explaining how the ranking model balances these often conflicting stages is an important consideration in a real world product.

Finally, the high level concept of "Guest Journey" applies beyond the Airbnb stays ranking (e.g., experience ranking, email marketing, etc). The main differences are in terms of product-specific milestones (e.g. negative stages such as unsubscription in email marketing). Crafting a model architecture that optimizes guest journey in Airbnb stays ranking, while also easily applied to other product use cases, is strategic to the success of the Airbnb ecosystem.

In this paper, we present **Journey Ranker**, a general multi-task model architecture with the following main contributions:

- **Learning both Positive and Negative Milestones:** Journey Ranker leverages intermediate guest actions to help the guest progress towards positive milestones, and avoid negative milestones throughout the guest journey.
- **Balancing Guest Journey:** Journey Ranker introduces a Combination module (Section 4.3) to balance the positive and negative milestones across the guest journey.
- **Modular and Extensible Model Architecture:** Journey Ranker is designed to consist of four modules with clear separation of concerns. This modular design allows it to be easily extended to other use cases beyond Airbnb stays ranking, which will be discussed in Section 5.6.

In this paper, we focus our presentation on a particular instantiation of the Journey Ranker for the Airbnb Stays ranking. Nevertheless, Journey Ranker is applicable to other search use cases where the user decision journey is long (e.g. e-commerce sites, real-estate listing sites, dating sites, etc), by choosing the appropriate milestones in the Base Module described in 4.1. Journey Ranker is also applicable when there are negative business outcomes (e.g. customer support tickets, poor product rating, product returns, etc) to be avoided, by modeling them in the Twiddler Module described in 4.2 and then balanced by the Combination Module described in 4.3. Journey Ranker could also be applied to other non-search journeys (e.g. email marketing, customer acquisition, etc) by configuring different positive and negative milestones. We present our successes in deployment in other use cases in Section 5.6.

The rest of the paper is organized as follows: In Section 2, we describe related work on search and recommendation systems. In Section 3, we present the Airbnb Stays Ranking problem formulation and the baseline. Next, we present Journey Ranker in Section 4 and the offline and online experiment results in Section 5. Finally, we discuss possible future directions in Section 6.

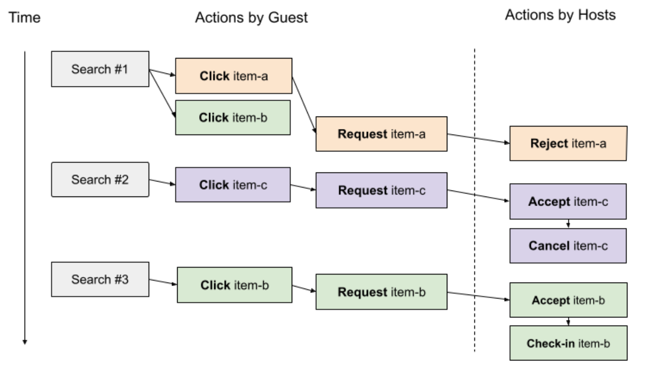


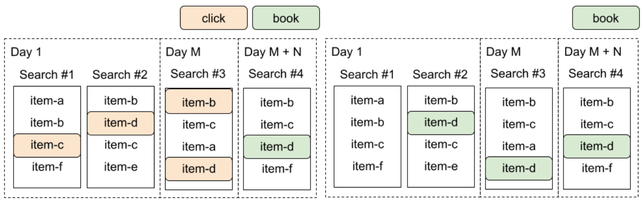
**Figure 1: An example Airbnb search journey for a guest who makes multiple searches, clicks, and reservation requests, before finally resulting in a realized stay.**

## 2 RELATED WORK

The majority of previous works in Search and Recommendation involves training deep learning models to optimize for a single business objective such as app installs in Mobile App Store [6], watch time in video recommendation [7], pCTR in advertisement [15], bookings of vacation rentals [8], etc. To deal with multiple potentially conflicting business objectives, one approach is to build an ecosystem of stand-alone models for each business objective [8], and apply them as "second-stage re-rankers" to finetune the top K results after the main ranker. In contrast, Journey Ranker incorporates all the different guest actions corresponding to various objectives in a single multi-task model trained end-to-end [4].

Recently, multi-task modeling has been an active topic in many Search and Recommendation use cases such as video recommendation [20], feed ads [11], etc., when there are multiple business metrics. Leveraging a multi-task approach allows sharing of feature representation, resulting in better performance for each task through transfer learning, while achieving lower overall serving cost. In contrast, for Airbnb Stays ranking, the main objective is uncancelled booking. Improving other metrics such as clicks without moving uncancelled booking implies a degraded ease of search experience for guests. Thus, our application of multi-task modeling is different from standard multi-objective settings in the sense that we are not trying to improve each of the objectives through shared representation. Rather, we leverage multi-task approach to let the model have a more comprehensive view of the guest search journey when optimizing for the uncancelled booking objective.

Such a strategy is similar to previous industry use cases that leverage multi-task modeling to model multi-step conversions, such as in e-commerce [12] and in customer acquisition [18]. Journey Ranker is inspired by these previous works, especially [12] in our design of Base Module in Section 4.1. In addition, we also extend previous work by adding new modules such as the Twiddler Module to model other negative search milestones and the Combination Module to better balance conflicting milestones throughout the guest search journey.



**(a) Example search journey** **(b) Example label attribution**

**Figure 2: An example Airbnb guest search journey and its label attribution.**

## 3 AIRBNB STAYS RANKING

### 3.1 Notations

To ease presentation, here are the notations we will use throughout the paper to describe an Airbnb user action, which will be used to denote an output, or a loss.

- *imp* - An impression of a search result in a search.
- *c* - A click of a search impression.
- *lc* - A long click of a search impression, defined by engagement of the listing details page after a click.
- *pp* - A click to open the payment page for a listing.
- *req* - A reservation request for a listing from a guest.
- *book* - An acceptance (by host) of a reservation request.
- *unc* - An uncancelled booking of a listing.
- *rej* - A rejection of a reservation request.
- *cbh* - A cancellation by host.
- *cbg* - A cancellation by guest.

We use *U*ppercase to denote a Set, *l*owercase to denote an instance of a Set, and subscript to denote relationship between two sets, for example:

- $U$ is a set of guests.
- $u$ is a guest within $U$.
- $S_u$ is a set of searches for guest $u$.

Finally, we use $y_t$, and $Loss_t$ to indicate the output, and loss for task (or module) $t$ respectively.

### 3.2 Previous Baseline

Figure 2 (left diagram) shows a simplified example of a typical guest search journey, where the guest makes multiple searches over multiple days and clicks on multiple search results to view their detail pages, before finally making a reservation request for a listing. The reservation request is then instantly-accepted if it is an Instant Book listing, otherwise the host manually decides whether to accept or reject the reservation request. Finally, both the guest and host may also cancel the booking before check-in. Thus, an **Uncancelled Booking** is one of the main metrics we use to evaluate the success of a guest search journey.

During model training of previous baseline, we label all of the search impressions of the booked listing in all of the searches the guest made before the booking occurred as "positive labels" as shown in Figure 2 (right diagram). The goal of the Airbnb Stays Ranking model is then formulated as ranking the positive labels higher than all the other search impressions; and we do this for all guest searches across the search journey so that guests can find their ideal listing as early as possible.

Specifically, we want to learn model parameters $\theta$ to minimize the loss across the searches, $S_u$, for all guests, $U$.

$$\underset{\theta}{\operatorname{argmin}} \sum_{u \in U} \sum_{s \in S_u} Loss_{unc}(s|\theta) \tag{1}$$

Note that the $Loss_{unc}$ is defined per search with the goal of ranking the search impressions associated with the booking higher than every other search impressions. The loss could be either listwise, or pairwise loss using standard learning-to-rank approaches [2] [3].

The previous production Airbnb Stays Ranking model then generates a score for each listing for ranking based on the model parameters $\theta$, listing features $F_L$, and context (i.e query and guest) features $F_C$ using a deep neural network consisting of two towers of fully connected layers with non-linear activation units [9] to predict uncancelled booking.

### 3.3 Disadvantages of Previous Baseline

While optimizing for booking aligns well with our business metric, there are a few disadvantages that are worth noting:

- We ignore searches from bookers without the booked listings, e.g. Search #1 in Figure 2 (right diagram).
- We equally treat all non-booked listings as negative examples, but in reality, some of them have clicks while others don't, e.g. *item-b* in Search #3 in Figure 2 (left diagram).
- We ignore all searches from non-bookers, including those guests who had reservation requests but were rejected by the hosts, which could provide useful insights for understanding guest and host preferences.

Given that both non-bookers and non-booking guest actions (eg. search clicks) are each one and two orders of magnitude bigger than bookers and booking events respectively, we are potentially missing out a significant amount of data to better extract guest and host preferences. Furthermore, we are also subject to *sample selection bias* problem [19] as we are serving a ranking model learned only from our past bookers to all our guests.

One important consideration for Airbnb is that conversion remains the ultimate goal. Specifically, we don't want a model that increases the number of clicks without increasing bookings, as that will imply that each booking involves more effort across a longer search journey to find the ideal listing.

Our previous attempts to remedy these issues included building a single multi-task model with booking and long click [8], and building multiple single-task models that are then combined online during serving using weights found through grid search. The former attempt increased long clicks but failed to move bookings. The latter attempt successfully increased bookings but had poor stability, resulting in out-of-date weights and a decline on on bookings a few months after the original launch.

In the next section, we present **Journey Ranker**, a novel multi-task model architecture trained end-to-end. Journey Ranker successfully drove significant business metric improvements without the downside of our previous approaches mentioned above.

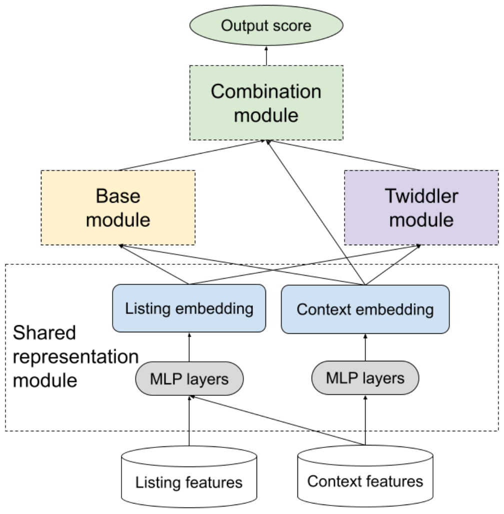


**Figure 3: A simplified view of the Journey Ranker model architecture, consisting of multiple modules - Shared Representation Module, Base Module, Twiddler module, and Combination module.**

## 4 JOURNEY RANKER FOR STAYS RANKING

Journey Ranker architecture consists of multiple modules trained end-to-end using multi-task techniques, and a simplified view of the architecture is shown in Figure 3 for the instantiation for our Airbnb Stays Ranking use case. The input to Journey Ranker are listing features $F_L$, and context (i.e query and guest) features $F_C$. Both $F_L$ and $F_C$ consist of both continuous and categorical features, and they are transformed accordingly through either normalization or embedding projection as described in previous work [8].

The Shared Representation Module takes both $F_L$ and $F_C$ as input and feeds them through multi-layer perceptrons (MLP) with non-linear activation units to generate listing embedding, $Emb_L$ and context embedding, $Emb_C$. Finally, $Emb_L$ and $Emb_C$ are fed to the three remaining modules (Base Module, Twiddler Module, and Combination Module) to generate the final output score. These modules will be presented in detail in the following subsections.

One thing to note is that if we remove the Twiddler Module and the Combination Module, and reduce the Base Module to be a single-task model optimizing for just uncancelled booking, then the output of the Base Module is equivalent to our previous production model architecture described in Section 3.2.

### 4.1 Base Module: Reaching Positive Milestones

The responsibility of the Base Module is to produce a single score per search result that is optimized for the final positive milestone (e.g. uncancelled booking for Airbnb stays ranking). As described in Section 3.3, we withhold a lot of information from the model if we simplify the complex Airbnb Stays ranking use case into a binary formulation of uncancelled booking vs search impression.

Thus, in Journey Ranker for Airbnb Stays Ranking, we reformulate the ML problem formulation for Base Module into helping guests reach each of the positive search milestones (i.e. improving the conversion funnel), culminating in an uncancelled booking. Motivated by [12], we also model the six positive guest search milestones as sequential guest actions that are linked together using the chain rule of probability. Specifically, we decompose the probability of uncancelled booking into each of the six guest milestones, i.e. Click *c*, Long Click *lc*, Payment page *pp*, Reservation request from guest *req*, Acceptance of reservation request by host *book*, and Uncancelled Booking *unc*, with the following equations:

$$\begin{aligned} P(unc) &= P(unc \mid book) * P(book) \\ P(book) &= P(book \mid req) \ * P(req) \\ P(req) &= P(req \mid pp) * P(pp) \\ P(pp) &= P(pp \mid lc) * P(lc) \\ P(lc) &= P(lc \mid c) * P(c) \end{aligned} \tag{2}$$

where each of the terms on the left hand side of the equation is actually a joint probability of all its preceding guest milestones, omitted for brevity. For example, $P(unc)$ is a joint probability, i.e. $P(unc \cap book \cap req \cap pp \cap lc \cap c \mid imp)$.

The instantiation of the Base Module for Airbnb Stays Ranking is in Figure 4, where for each of the guest milestones, we generate a corresponding task logit (i.e. blue box) by running a multi-layer perceptron (MLP) on top of the Listing Embedding $Emb_L$ and Context Embedding $Emb_C$ from the Shared Representation Module. In other words, the six blue boxes in Figure 4 correspond to the six conditional probabilities defined in Equation 2.

To train the Base Module for the Journey Ranker, we apply multiple losses, one for each of the joint probabilities in Equation 2 as shown in the Figure 4. For example, for a specific task such as payment page, all the search impressions that led to a payment page view are considered positives and the rest are considered negatives.

The final loss for the Base Module is the sum of each of the losses from each individual tasks:

$$Loss_{Base} = \sum_{task} Loss_{task} \tag{3}$$

where, using the same notation as Equation 1, $Loss_{task}$ is:

$$Loss_{task} = \sum_{u \in U} \sum_{s \in S_u} loss_{task}(s|\theta) * w_{task} \tag{4}$$

where $loss_{task}(s|\theta)$ corresponds to loss for a given search $s$ computed using standard learning-to-rank approaches [2] [3], and $w_{task}$ is a normalization term for a given task to avoid tasks with a lot of occurrences (e.g. clicks) from dominating the final loss.

E.g., suppose for the long click task, we use softmax listwise loss for $loss_{lc}(s|\theta)$, where each search that consists of N long clicks will result in N softmax loss. In that case, $w_{lc}$ will be the empirical fraction of long clicks that will convert into an uncancelled booking in our training data. The intention behind designing such normalization term is:

- Each positive milestone is assigned credit proportional to its likelihood to convert into the final uncancelled booking.
- The Base Module is learning multiple different but related reformulations of an uncancelled booking (i.e. through intermediate search milestones), which could help the model learn a better Shared Representation Module.

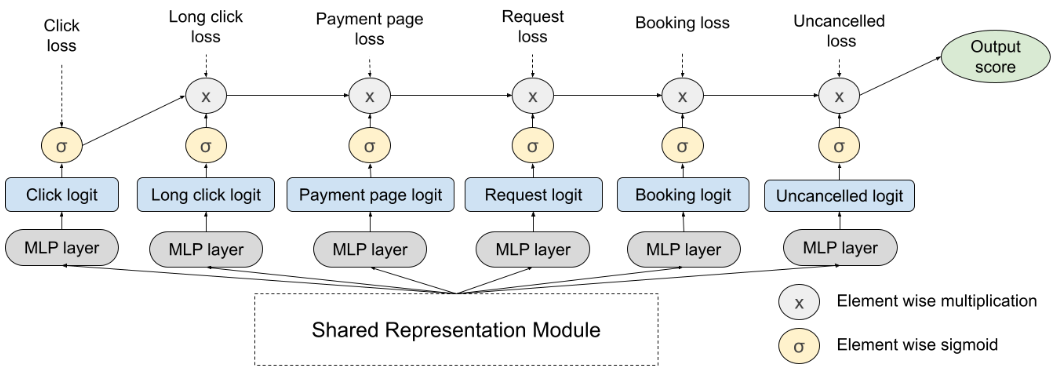


**Figure 4: Airbnb Stays Ranking's Instantiation of Base Module.**

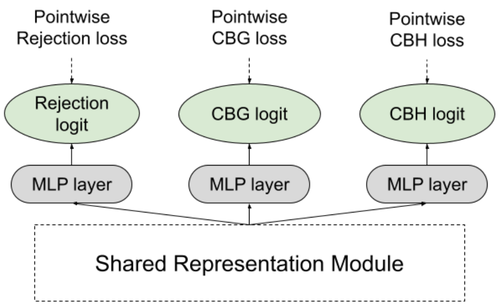


**Figure 5: Airbnb Stays Ranking's Instantiation for the Twiddler Module.**

Overall, the design of Base Module as described above allows us to better capture the sparse, long-duration, exploratory, and assorted guest activities at Airbnb, which is an important challenge for the Airbnb product use case, as described in the Section 1. Note that other use cases could have different number of positive milestones corresponding to their specific guest journey.

## 4.2 Twiddler Module: Avoiding Negative Milestones

The responsibility of the Twiddler Module is to produce a score per search result for each of the negative milestones. The instantiation of Twiddler Module for Airbnb Stays Ranking can be found in Figure 5, where we have three Twiddler tasks, i.e. rejection, cancellation by host, and cancellation by guest. Twiddler Module is learned in the same multi-task setting with each of the Twiddler tasks receiving as input the embeddings from the Shared Representation Module, and passing the embeddings through a multi-layer perceptron (MLP) to generate the final logit. Similar to Equation 3, the loss for Twiddler Module is just the sum of each of the twiddler losses:

$$Loss_{Twiddler} = \sum_{task} Loss_{task} \tag{5}$$

where $Loss_{task}$ is a binary classification loss for each of the negative milestones (eg. rejected reservation request vs accepted reservation request for the Rejection Twiddler task).

One might wonder why we can't reuse the corresponding conditional probabilities (eg. $P(book \mid req)$) in Equation 2 from the Base Module in Section 4.1 for the twiddlers here. The reasons are:

- In Base Module, we apply the losses directly on the joint probabilities instead of the conditional probabilities. Thus, the conditional probabilities are biased and not accurate [14].
- We achieve more accurate modeling of cancellations by separately modeling cancellations initiated by the guest and host. However, since the Base Module currently uses a linear multi-step conversion setup, it cannot implement this separation (i.e. requires a "forked" conversion funnel instead of a linear conversion funnel).
- These negative search milestones are rarer (<1% to 10% depending on the action), leading to class imbalance. They are also delayed outcomes where the final label could arrive after a long time has elapsed. By designing them as its own separate module, we can optimize each of the twiddlers further to account for its unique circumstances such as different sampling strategies, etc. in future works.

Overall, the Twiddler Module allows us to incorporate negative milestones from both guest and host in the search journey, which is one of the unique challenges for Airbnb product described in Section 1. Note that other use cases would have different twiddlers, corresponding to different use case specific negative milestones.

## 4.3 Combination Module: Balancing Guest Journey

Finally, the responsibility of the Combination Module is to balance the output from the Base Module that is optimized for positive milestones and the outputs from the Twiddler Module that are optimized for negative milestones. Figure 6 shows the Airbnb Stays Ranking's instantiation for the Combination Module, which can be represented as the following:

$$y_{Combination} = y_{Base} * \alpha_{Base} + \sum_{t \in twiddler} y_t * \alpha_t \tag{6}$$

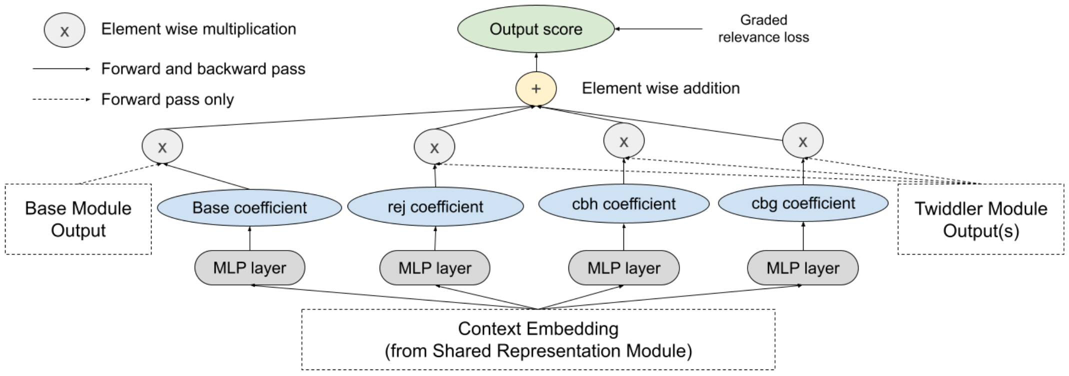


**Figure 6: Airbnb Stays Ranking's Instantiation for the Combination Module.**

where the final output score, $y_{Combination}$, is obtained by applying a linear combination on top of the Base Module score, $y_{Base}$, and the Twiddler Module scores, $y_t$. Note that each coefficient $\alpha_x$ in the linear combination is a scalar learned end-to-end in the Combination Module, generated using a MLP that takes in the Context Embedding $Emb_C$ from the Shared Representation Module.

There were several key design choices made, i.e.:

- **A context-dependent coefficient $\alpha$ instead of a global coefficient:** As shown in Figure 1, the Airbnb search journey is very long and the needs of the guest change as they gather more information along the journey. E.g., if a guest already experienced a negative search milestone in the journey, we want to emphasize avoiding another negative search milestone to prevent churn. A context-dependent coefficient allows the model to learn to balance the guest search journey better as we know more about the guest.
- **A linear combination instead of a non-linear combination of the various outputs:** We attempted both options and they perform similarly in the offline evaluation. Thus, we opted for the simpler linear combination manner as it allows better product interpretability, which will be presented in more details in Section 5.4.
- **Freezing the gradient update from Combination Module to the Base and Twiddler Module:** Note that in the Figure 6, we do not have backward pass from the Combination Module to the output from the Base Module and Twiddler Module. We designed it this way so that the Combination Module can focus merely on learning how to combine these outputs without changing their semantics.

Such designs allow us to better understand what the module is learning, which is explored further in Section 5.4.

Finally, we devise an objective function for the Combination Module so that it learns to properly balance positive and negative search milestones. To do that, we construct a pairwise loss for the final output of the Combination Module to learn graded relevance of different guest actions based on the following pairwise relationship:

$$unc > click > imp > cbg \,/\, cbh \,/\, rej \tag{7}$$

Specifically, uncancelled booking is considered the best outcome, followed by clicks and then all of the unclicked search impressions. All the negative search milestones (i.e. cancellation by guest, cancellation by host, and rejection) are considered the worst outcomes, even worse than the unclicked search impressions.

Finally, the full loss is the sum of losses across all modules, i.e.

$$Loss_{Total} = Loss_{Base} + Loss_{Twiddler} + Loss_{Combination} \tag{8}$$

Note that other use cases could define Equation 7 differently to optimize for their own version of a satisfied user (e.g. good product rating without product returns in e-commerce sites).

## 4.4 Richer Training Data

To enable Journey Ranker to learn from various different guest actions, we make the following changes to the training data:

- **Labeling of Search Impressions:** We label each search impression with all the labels that are applicable (i.e. multi-label) - both positive guest actions (i.e. click, long click, payment page, reservation request, accepted booking, and uncancelled booking) and negative guest actions (i.e. rejection, cancellation by guest, and cancellation by host).
- **Number of Searches:** By leveraging guest actions such as clicks, we can now leverage two orders of magnitude more data for our model training. However, this introduces both scalability complexity and more noise to our model training due to lower intent exploratory guests. In the end, we included only all the searches that led to a payment page view after evaluating empirically the trade off between number of searches and the model performance. This leads to just around 50% more searches compared to before.

## 5 EXPERIMENTAL RESULT

For offline evaluation, we use Normalized Discounted Cumulative Gain (NDCG) with binary relevance score, where uncancelled booking is assigned a relevance score of 1 and all other search impressions are assigned a relevance score of 0.

For model training, we use around 500 millions searches for training, and all the models are trained using Tensorflow. All the models are trained with five different initializations and we report the average offline performance along with its 95% confidence interval. Both the training throughput and the end-to-end model latency are similar compared to the previous baseline, as we are only adding +9.2% parameters.

| Model | Offline NDCG | # Parameters |
|---|---|---|
| Baseline | +0.0% (±0.03%) | +0.0% |
| Journey Ranker | +0.48% (±0.05%) | +9.2% |

**Table 1: End-to-end offline evaluation for the Airbnb Stays' instantiation of Journey Ranker described in Section 4.**

## 5.1 End-to-end Offline Evaluation

Table 1 shows the offline evaluation of the Journey Ranker described in Section 4 compared to the previous Baseline model described in Section 3.2. Overall, Journey Ranker has a statistically significant improvement of +0.48% NDCG compared to the baseline, with a small 9.2% increase in number of trainable parameters. The offline result is encouraging because for Airbnb Stays ranking use case:

- An offline evaluation of +0.3% is considered promising to land a statistically significant booking gain in online A/B test, which we will present in Section 5.5 later.
- For the baseline Airbnb Stays Ranking, a +50% in trainable parameters (through making the network deeper or wider) will only have about 0.05-0.1% offline NDCG gain.
- Despite having a more complicated model architecture (10 tasks!), we only incur a small 9.2% increase in number of trainable parameters because each of the new task-specific networks is a small (single-layer) network that shares the same representation layers before it. This means we still have more opportunity to improve gain further by increasing the capacity for each task using other techniques [13] [16].

In the following subsections, we will examine in more detail how each of the Modules within Journey Ranker behaves to shed some light on where the improvements come from.

## 5.2 Discussion of Alternative Designs

In this section, we describe some of the alternative designs that we attempted and present our findings.

**A single end-to-end module** - Could we design a single task model that directly optimizes for graded relevance described in the Equation 7? We attempted this and the naive application of the same graded relevance resulted in > 1% drop in offline NDCG. We can get a better result (but still worse than Journey Ranker's) if we tune the relative weights between each of the labels in the graded relevance setup. We hypothesize that by giving the model full freedom, the different labels will interfere with each other to compete for the same model parameters, causing the more prevalent tasks (eg. click) to dominate the final output. By designing the model architecture using well-defined modules and tasks with separation of concerns, each of the tasks will have its own task-specific parameters to learn knowledge specific to itself, while contributing to the shared representation layer for the common knowledge.

**Non-uniform loss weights** - This motivates another question of whether we should tune the relative weights of losses for each of the tasks in Journey Ranker? Note that, as described in Equation 8, Journey Ranker currently uses a simple unweighted sum of losses across each of the modules (and thus each of the tasks). Interestingly, we tried to manually tune these weights but did not achieve very statistically significant offline results that could justify the increase in complexity. However, we believe there is opportunity in the future to revisit using an auto-hyperparameter tuning service or other surrogate tricks [11] to improve this further.

| Tasks Setting | Offline NDCG | Params | Searches |
|---|---|---|---|
| Baseline (Unc) | +0% (±0.03%) | +0.0% | +0% |
| Req + Book + Unc | +0.06% (±0.03%) | +1.8% | +25% |
| Click + Unc | +0.22% (±0.02%) | +1.8% | +50% |
| All 6 Tasks | +0.31% (±0.02%) | +6.0% | +50% |

**Table 2: Ablation study on the tasks used in the Base Module for Airbnb Stays Ranking, without Twiddler and Combination Module. We present four settings that have both practical product semantics in Airbnb Stays Ranking and also stat-sig difference in offline NDCG across each pairs.**

**Dealing with negative transfer** - We also tried various previous works on dealing with gradient interference [5] and multi-task loss weighting [10] for Journey Ranker and did not see stat-sig improvement in offline NDCG. We hypothesize that this is because all of the ten tasks we leverage in Journey Ranker are ten different aspects of understanding the same guest preference (i.e. uncancelled booking), and thus don't suffer from standard multi-task challenges where the tasks are very different from each other.

## 5.3 Can we simplify the Base Module?

In the Base Module, we have six tasks (clicks, long clicks, payment page, requests, accepted bookings, and uncancelled bookings) corresponding to six different search milestones in the guest search journey. One might wonder whether we need all six of these?

Table 2 shows the ablation study of the tasks used in the Base Module, without the Twiddler and Combination Module to better understand the effect. We presented four choices of tasks that have useful product semantics in Airbnb and also statistically significant improvement in offline NDCG across each pair in the setup. There are a few useful learnings:

- **Quantity of searches:** As expected, as the number of searches used for training the Base Module increases, the offline NDCG improves (i.e. from row 1 to 3 in Table 2).
- **"Diversity" of searches:** Adding new searches that are more different from existing searches could lead to more gain. E.g., in Table 2, adding +25% more searches with reservation requests (that was ultimately rejected or cancelled) only had a +0.06% NDCG improvement (row 2 vs row 1). In contrast, if we add +20% more searches with clickers (that didn't have any reservation requests), we had a +0.16% NDCG improvement (row 3 vs row 2). This aligns with our hypothesis that by only utilizing searches from uncancelled bookers, we might not generalize well to the non-bookers, who might have very different guest behavior.
- **Usefulness of intermediate search milestones:** In table 2, if we compare row 4 vs row 3, we have a +0.09% offline NDCG improvement despite having the same number of searches. This suggests that adding more intermediate search milestone labels to better differentiate the guest preference on each of the search impressions is useful. In the future, we plan to try other efforts to maximize the signals derived from each search, such as through different model architectures for Base Module [1] [17].

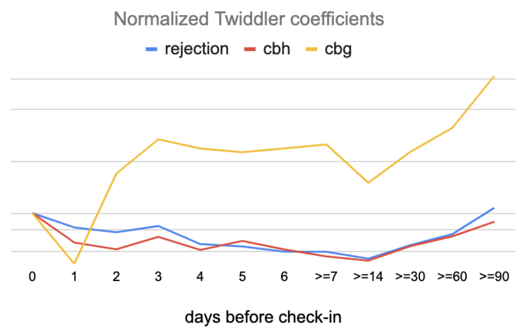


**Figure 7: NTC when segmented by days ahead of the check-in for a search query. Each NTC curve is normalized by NTC when days before check-in = 0 for ease of visualization.**

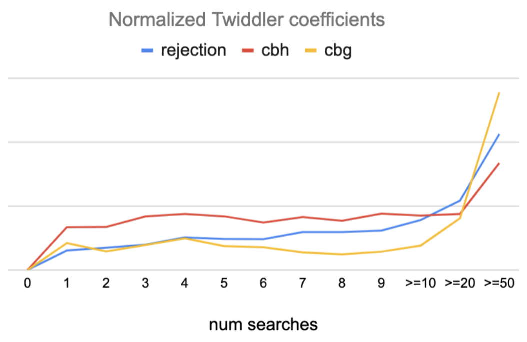


**Figure 8: NTC when segmented by guest's number of previous searches. Each NTC curve is normalized by NTC when num searches = 0 for ease of visualization.**

In addition to offline NDCG for the uncancelled booking as the label, we also look at the offline NDCG for all the other labels such as clicks, long clicks, payment pages, and requests. We omit the detailed result for brevity, and the high level observation is that the offline NDCG for each of these search milestones also has a statistically significant gain. This gives another insight on where the offline NDCG gain for uncancelled booking comes from - by improving conversion for each of the intermediate search milestones, the final uncancelled booking will also naturally increase.

## 5.4 What does the Combination Module learn?

As described in Equation 6, the output of the Combination Module is a linear combination of the outputs from the Base Module and the Twiddler Module. Recall that each coefficient is learned end-to-end based on the Context Embedding $Emb_C$, and thus are dependent on Context Features. This allows us to study how these coefficients change given different guest or query features to try to decipher what the Combination Module learns.

**Definition** - We compute the "*Normalized Twiddler coefficient*" (NTC), which normalizes the coefficient for each of the Twiddler outputs, $\alpha_{twiddler}$ against the coefficient of the Base Module, $\alpha_{base}$. We can now plot a diagram for each context feature, where the x-axis is the various values for a particular feature, and the y-axis is the NTC metric to understand how the NTC changes as the feature value changes. Note that a higher NTC indicates that the Combination Module is giving more importance to a particular Twiddler task, i.e. more importance to avoiding a specific negative search milestone, compared to the Base Module.

Next, we present examples of what the Combination Module learns with respect to two context features:

**An example query feature** - Figure 7 shows how NTC moves with respect to the query feature "days ahead of check-in". Here, we notice that there are two different shapes for the NTCs:

- Rejection and cancellation by host are both negative search milestones initiated by the host, and thus it makes sense that both of their NTC curves are similar - i.e. a U-shaped curve. This shape aligns with our product intuitions for when the search product should try to minimize rejections or cancellation by hosts with respect to this query feature. Specifically, if the trip is close to the query time, there is not enough time for guests to rebook. On the other hand, if the trip is far from the query time, minimizing the likelihood of future unforeseen circumstances that cause the host to to reject / cancel before the trip is crucial.
- We hypothesize that the NTC curve for cancellation by guest is increasing for trips that are further out to prevent the guest from cancelling due to buyer's remorse (eg. found a better listing or realizing that the listing does not satisfy their need due to location or missing amenities etc).

We hypothesize that the model learns these NTC curves through a mixture of reasons, such as:

- Differences in prevalence of the negative search milestones for each query segment.
- Label attribution in training data - E.g., if the trip is coming up soon, the guest might not have time to rebook on Airbnb or might have opted for other options such as a traditional hotel. On the other hand, if the trip is far into the future, the guest still has plenty of time to rebook. Since we use a max window of K days to define a guest search journey, the successive searches might be grouped into a separate search journey. Regardless of the cases, the original search journey is now considered a failed journey, and the model learns to try to avoid these negative outcomes.

**An example guest feature** - Figure 8 shows how NTC varies against the guest's number of previous searches. We observe that as the guest conducts more searches, the NTC increases. This implies that the model is focusing more on avoiding the negative search milestones as the guest is further into their guest journey.

To understand this further, we analyzed the Pearson correlation coefficients between the historical click-through-rate of a listing and the rejection rate & cancellation rate of a listing and noticed that the correlations are positive. Recall that click is modeled as a positive search milestone in the Base Module while the rejection and cancellations are modeled as negative search milestones in the Twiddler Module. Thus, a higher NTC for a Twiddler task implies the model is trying to avoid the corresponding negative outcome, and thus indirectly aiming for a lower click-through rate due to the positive correlation between the two.

In other words, what the model learned aligns with our product intuition. Specifically, we want the guest to explore more in the

| Airbnb Product | Uncancelled Bookers | Clickers |
|---|---|---|
| Stays | +0.61% | +0.14% |
| (In-real-life) Experiences | +2.0% | +1.1% |
| Online Experiences | +9.0% | +1.3% |

**Table 3: Online experiment results across different Airbnb Search Products, all gains have p-values < 0.01**

beginning of the search journey, by showing listings with high click-through rate, even though they could lead to negative outcomes. As the guest narrows down to the final decision, we want to rank higher the listings that will not result in rejection or cancellation.

We hypothesize that the model manages to learn such information exactly due to the different guest behaviors displayed across the search journey. E.g., during the early searches, the guest normally clicks more to better understand the inventory, but narrows down to just a few choices in the latter part of the journey.

**Summary** - We found that what the Combination Module learns aligns well with our product intuitions in how we might try to balance the guest search journey across guest state and query segment. One might also wonder whether each of the tasks (eg. Base Module output or the three outputs from the Twiddler Module) might already pick up such information through their output score. We plot similar curves for the outputs, $y_t$ from Equation 6 in the Combination Module with respect to the same context features and observe that the curves are flat. We hypothesize that the model was unable (or unnecessary) to learn such journey-level information because it has access to both the context features, $F_C$, and the listing features, $F_L$, during the model training. Since the listing features, $F_L$ are likely more predictive of the negative search milestones, it dominates the final prediction. By using just the Context Embedding, $Emb_C$, to learn the coefficients in the Combination Module, we essentially guide the module to focus on how these context features could affect the various negative search milestones.

Overall, the ability to interpret what the model learns is extremely helpful from the product perspective, and this helps us to tackle the challenges that we highlighted in Section 1.

### 5.5 Online Experiments for Stays Ranking

As shown in Row 1 of Table 3, Journey Ranker drove +0.61% gain in uncancelled bookers compared to the baseline when tested in online A/B experiment. We also saw strong gain throughout the whole conversion funnel, such as an increase of +0.14% from searchers to clickers, and an increase of +0.48% from clickers to uncancelled bookers, which aligns with our observation in the offline evaluation.

### 5.6 Online Experiments Beyond Stays Ranking

In addition to stays ranking, we also successfully applied the Journey Ranker architecture to other search use cases in Airbnb. E.g., row 2 and row 3 in Table 3 show +2.0% bookers for experiences ranking and +9.0% bookers for online experiences ranking. Note that the absolute metric gains for both of these use cases are much higher compared to the Airbnb stays ranking. This is expected since the newer products have a lot less bookings for the model to learn from. Thus, by considering the full guest search journey, we are able to add more incremental value towards better learning guest preference. This also implies that Journey Ranker is most useful for the cold start problem where the absolute number of conversions is smaller, leading to model underfitting, allowing us to more easily ramp up new businesses in the future.

It is worth noting that the implementation for the Experiences use cases is a trimmed down version (i.e. less tasks) of the Airbnb Stays' instantiation described before. This is because the Experiences product lacks reliable telemetry for some intermediate guest actions described. Experiences product also does not have rejection outcomes since all experiences are auto-accepted. Based on offline results for Airbnb stays ranking presented in Table 1 and Table 2, we hypothesize that we can achieve even more gain for the Experiences product once reliable telemetry is added for all of the experience guest actions throughout the search journey.

We also successfully applied the Journey Ranker architecture in Airbnb beyond these search applications. E.g., we collaborated with the Airbnb Email marketing team to leverage the Journey Ranker architecture for the email recommendation use case and achieved +0.7% nights booked and -3.7% email unsubscribes.

This is the first modeling effort within Airbnb to achieve significant business gains across multiple Airbnb products with minimal customization thanks to the modular and extensible model architecture. The main considerations needed to leverage this architecture is grouping the guest actions as either multi-step positive milestones in the Base Module or negative milestones in the Twiddler Module and then defining a graded relevance to balance the two.

## 6 CONCLUSION AND FUTURE WORK

We presented Journey Ranker, a novel multi-task learning framework for Airbnb Stays ranking that balances positive and negative milestones throughout the guest search journey. Journey Ranker significantly outperforms the previous baseline in both offline and online evaluation, and we presented some of the offline study to provide insight into what the model learns. We also saw great results applying Journey Ranker architecture in three other search and non-search use cases in Airbnb. For future work, we believe that other research ideas could easily fit in the Journey Ranker architecture. E.g., we could extend the Base Module with even more micro-tasks that don't follow a sequential multi-step conversion through graph convolution networks [1]. We could also easily extend the Twiddler Module with even more negative milestones such as customer support tickets, trip quality etc. Finally, we can modify the Combination Module to directly optimize for business Overall Evaluation Criteria (OEC) and weigh various outcomes using Future Incremental Value (FIV) estimations.

## ACKNOWLEDGMENTS

No journey (pun intended) is possible without contributions from a lot of people. Specifically, we would like to thank Francisco Selame and Bo Yu for collaborating on the Airbnb Experiences use case; and Teng Wang for the Airbnb Email Marketing use case. We would also like to extend our thanks to Alex Deng, Michelle Du, Anna Matlin, and Tatiana Xifara for their help in the online experiment analysis to enable the deployment of the Journey Ranker into production across multiple Airbnb product use cases. Finally, we would like to thank the entire Airbnb Relevance team for valuable discussions.

## REFERENCES

[1] Wentian Bao, Hong Wen, Sha Li, Xiao-Yang Liu, Quan Lin, and Keping Yang. 2020. GMCM: Graph-Based Micro-Behavior Conversion Model for Post-Click Conversion Rate Estimation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 2201–2210. https://doi.org/10.1145/3397271.3401425

[2] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[3] 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)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[4] Rich Caruana. 2004. Multitask Learning. *Machine Learning* 28 (2004), 41–75.

[5] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2017. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. (2017). https://doi.org/10.48550/ARXIV.1711.02257

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (Boston, MA, USA) *(DLRS 2016)*. Association for Computing Machinery, New York, NY, USA, 7–10. https://doi.org/10.1145/2988450.2988454

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[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*. ACM. https://doi.org/10.1145/3292500.3330658

[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)*. Association for Computing Machinery, New York, NY, USA, 2822–2830. https://doi.org/10.1145/3394486.3403333

[10] Alex Kendall, Yarin Gal, and Roberto Cipolla. 2017. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. https://doi.org/10.48550/ARXIV.1705.07115

[11] Ning Ma, Mustafa Ispir, Yuan Li, Yongpeng Yang, Zhe Chen, Derek Zhiyuan Cheng, Lan Nie, and Kishor Barman. 2022. An Online Multi-Task Learning Framework for Google Feed Ads Auction Models. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 3477–3485. https://doi.org/10.1145/3534678.3539055

[12] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. 2018. Entire Space Multi-Task Model: An Effective Approach for Estimating Post-Click Conversion Rate. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 1137–1140. https://doi.org/10.1145/3209978.3210104

[13] Elliot Meyerson and Risto Miikkulainen. 2018. Pseudo-task Augmentation: From Deep Multitask Learning to Intratask Sharing - and Back. In *ICML*.

[14] Hao Wang, Tai-Wei Chang, Tianqiao Liu, Jianmin Huang, Zhichao Chen, Chao Yu, Ruopeng Li, and Wei Chu. 2022. ESCM2: Entire Space Counterfactual Multi-Task Model for Post-Click Conversion Rate Estimation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 363–372. https://doi.org/10.1145/3477495.3531972

[15] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17* (Halifax, NS, Canada) *(ADKDD'17)*. Association for Computing Machinery, New York, NY, USA, Article 12, 7 pages. https://doi.org/10.1145/3124749.3124754

[16] Yuyan Wang, Zhe Zhao, Bo Dai, Christopher Fifty, Dong Lin, Lichan Hong, Li Wei, and Ed H. Chi. 2022. Can Small Heads Help? Understanding and Improving Multi-Task Generalization. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 3009–3019. https://doi.org/10.1145/3485447.3512021

[17] Hong Wen, Jing Zhang, Yuan Wang, Fuyu Lv, Wentian Bao, Quan Lin, and Keping Yang. 2020. Entire Space Multi-Task Modeling via Post-Click Behavior Decomposition for Conversion Rate Prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 2377–2386. https://doi.org/10.1145/3397271.3401443

[18] Dongbo Xi, Zhen Zi Chen, Peng Yan, Yinger Zhang, Yongchun Zhu, Fuzhen Zhuang, and Yu Chen. 2021. Modeling the Sequential Dependence among Audience Multi-step Conversions with Multi-task Learning in Targeted Display Advertising. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021).

[19] Bianca Zadrozny. 2004. Learning and Evaluating Classifiers under Sample Selection Bias. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 114. https://doi.org/10.1145/1015330.1015425

[20] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending What Video to Watch next: A Multitask Ranking System. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997