RankFormer: Listwise Learning-to-Rank Using Listwide Labels

Maarten Buyl
Amazon
Luxembourg, Luxembourg
Ghent University
Ghent, Belgium
maarten.buyl@ugent.be

Paul Missault
Amazon
Luxembourg, Luxembourg
pmissaul@amazon.com

Pierre-Antoine Sondag
Amazon
Luxembourg, Luxembourg
pierreas@amazon.com

ABSTRACT

Web applications where users are presented with a limited selection of items have long employed ranking models to put the most relevant results first. Any feedback received from users is typically assumed to reflect a relative judgement on the utility of items, e.g. a user clicking on an item only implies it is better than items not clicked in the same ranked list. Hence, the objectives optimized in Learning-to-Rank (LTR) tend to be pairwise or listwise.

Yet, by only viewing feedback as relative, we neglect the user's absolute feedback on the list's overall quality, e.g. when no items in the selection are clicked. We thus reconsider the standard LTR paradigm and argue the benefits of learning from this listwide signal. To this end, we propose the RankFormer as an architecture that, with a Transformer at its core, can jointly optimize a novel listwide assessment objective and a traditional listwise LTR objective.

We simulate implicit feedback on public datasets and observe that the RankFormer succeeds in benefitting from listwide signals. Additionally, we conduct experiments in e-commerce on Amazon Search data and find the RankFormer to be superior to all baselines offline. An online experiment shows that knowledge distillation can be used to find immediate practical use for the RankFormer.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; • **Computing methodologies** → **Learning from implicit feedback**.

KEYWORDS

learning-to-rank, transformer, listwise ranking, listwide, attention

ACM Reference Format:

Maarten Buyl, Paul Missault, and Pierre-Antoine Sondag. 2023. RankFormer: Listwise Learning-to-Rank Using Listwide Labels. 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, 12 pages. https://doi.org/10.1145/3580305.3599892

1 INTRODUCTION

Ranking problems naturally arise in applications where a limited selection of items is presented to a user. A classic example is web search engines [11], which show the most relevant URLs to a query. Similarly, e-commerce websites want to show products that customers want to buy [40] and streaming services show content that they want to watch [14]. The user wants to find the item that offers the most utility, as quickly as possible. The selection is thus shown in an ordering, e.g. a list, where the 'best' items are shown first.

The field of Learning-to-Rank (LTR) is well-established in machine learning as a way to form these rankings based on user feedback on previous lists [7, 10, 25]. In many real-world applications, this feedback is implicit: it is collected indirectly from user behavior. For example, if users often purchase a certain product when shown in response to a certain query, then that product should probably be ranked highly in future responses to that query.

Traditionally, the models behind LTR are simple regressors, such as Gradient-Boosted Decision Trees (GBDTs) [13] or neural networks [9], that compute a *pointwise* score, i.e. a score that is independently determined for every item in a list. Yet it is a fundamental aspect of LTR that any feedback about particular items should be seen in relation to feedback to other items in the list, since user feedback tends to reflect *relative* utility judgments rather than *absolute* judgments [19]. Hence, it is misleading to make *pointwise* comparisons between a ranking model's score for an item and its feedback label. Rather, *pairwise* [7] objectives are used that compare pairs of scores to pairs of labels, or *listwise* [10] objectives that jointly compare scores to labels for the entire list.

However, a critical shortcoming of pairwise and listwise approaches is that they *only* consider user feedback as a relative judgement. In particular, lists that did not receive any feedback tend to be ignored. Yet this absolute judgement of lists should not be disregarded. In e-commerce, lists that resulted in many purchases are likely to have an overall better selection than lists without any user interaction. In the latter case, the overall selection may be poor for many reasons: because the user gave a bad search query, because the selection of products is too expensive, or because the user is looking for new items that have seen little interaction so far.

Contributions.

(1) We make the case for learning from *listwide* labels, i.e. absolute judgments of a list's overall quality, and formalize an approach to derive them as the list's top individual label.
(2) We propose the *RankFormer* architecture, illustrated in Fig. 1. At its core, the RankFormer utilizes the *listwise* Transformer [38] architecture that was recently applied to perform listwise ranking [30, 34, 37]. It combines this listwise LTR objective with a listwide objective that performs an overall quality assessment of the list and compares it to the listwide labels.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599892

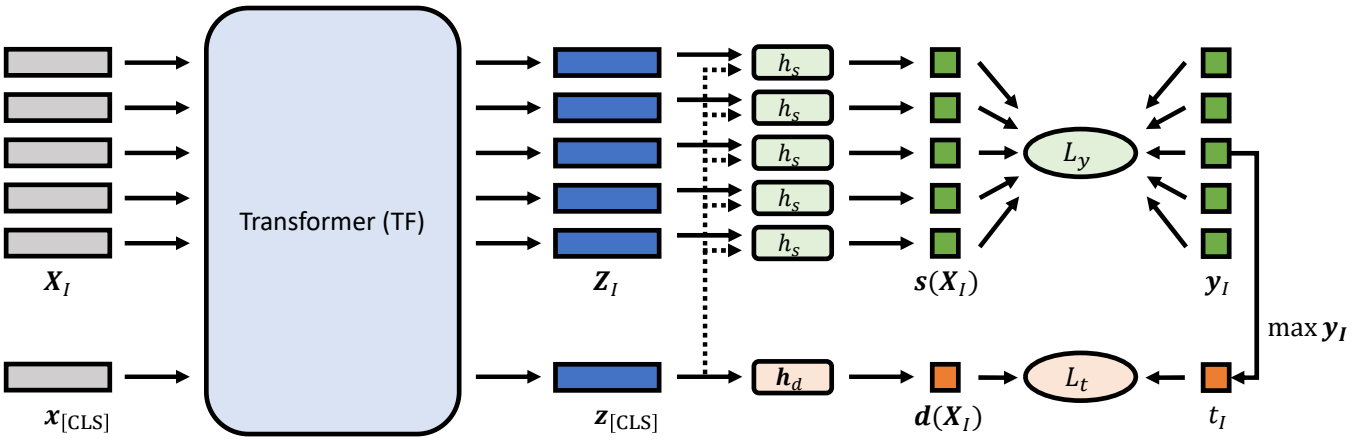

Figure 1: The RankFormer architecture. Features X_I of selection I, accompanied by the `[CLS]` token vector $x_{[CLS]}$, are sent through the listwise Transformer TF. The returned features Z_I each correspond with an item $i \in I$, whereas the returned $z_{[CLS]}$ vector models the overall list quality. The transformed vectors z_i of individual items $i \in I$ are each concatenated with $z_{[CLS]}$ and sent through a shared scoring head h_s to obtain scores $s(X_I)$, which are optimized with a list*wise* LTR loss L_y based on the item feedback labels y_I. Additionally, $z_{[CLS]}$ is sent through a separate head h_d to generate the list quality prediction $d(X_I)$, which should match the t_I, i.e. the highest item label in the list, and is optimized with a list*wide* loss L_t.

Thus, the RankFormer explicitly models the overall list quality and generates individual item ranking scores relative to that overall quality.

(3) A key use case for learning from listwide objectives is to learn from lists that were presented to users but received no implicit feedback. Yet, such feedback is not provided in the publicly available datasets that are popular in LTR benchmarks, where only *explicit* feedback is given. We therefore simulate implicit feedback on public datasets and observe that the RankFormer successfully leverages its listwide objective to improve upon other neural ranking methods.

(4) We conduct experiments on data collected from Amazon Search in both an offline and online setting. Offline, the RankFormer is superior to all baselines. Online, this advantage can be maintained when distilling the RankFormer to a simple GBDT model. Despite its complexity, the RankFormer is thus readily put into practice.

2 RELATED WORK

Learning-to-Rank (LTR) provides tools, e.g. objective functions, to learn from feedback labels such that a ranking model is improved. To this end, *neural* ranking uses neural networks as the model.

A distinction is made with (neural) *Information Retrieval* [17], which often applies (neural) ranking to obtain information related to a query from a single collection of information resources. Though that field aims to find the *most* relevant resources, it has less of a focus on the LTR objectives themselves and more so on the relevance of individual information resources. Instead, we assume that a limited selection of relevant items has already been made, and our model should now put the most relevant of these on top.

Traditional LTR [25] employs a *pointwise* scoring function, i.e. one that computes an individual score for each item independently, and only then compares those scores in pairwise [7] or listwise [10] loss functions. Yet the actual utility of items may depend on the rest of the list, e.g. an item may be much more desireable if it is the cheapest in the selection. Methods have therefore been proposed to compute *listwise* scores directly, i.e. scores that depend on the features of all items in the list. To this end, Ai et al. [1] proposed the Deep Listwise Context Model, which uses a recurrent neural network to take other inputs into account when reranking the top documents in information retrieval. This approach was further formalized to general multivariate scoring functions [3].

A list of items can be seen as a sequence, hence many approaches from Natural Language Processing (NLP) can readily be applied to ranking. Most notably, the Transformer [38] architecture has been used to learn from listwise ranking contexts using self-attention, which we refer to as *listwise* Transformers [24, 30–32, 34, 37, 42]. As a sequence-to-sequence architecture, the Transformer here considers sequences of items instead of words. This form of *listwise* attention is thus orthogonal to any usage Transformers that may be used to compute the feature vectors of individual items.

Particular to our approach, the RankFormer, is that we use the listwise Transformer to combine the listwise LTR objective with a *listwide* objective, for which the overall quality of the list is explicitly modelled. The Deep Listwise Context Model proposed by Ai et al. [1] also models the list context with a recurrent neural network, yet this model is not used to actually predict listwide signals. The most related work to ours is the Personalized Re-ranking with Contextualized Transformer for Recommendation (PEAR), recently proposed by Li et al. [24]. In the personalized recommendation setting, the PEAR model combines a pointwise ranking objective, i.e. predicting whether the item is clicked, with a listwide objective where it should be predicted whether *any* item is clicked. Our work formalizes the listwide objective as an ordinal loss and combines it with a listwise ranking objective in the general LTR context.

In recent work on score calibration in LTR, Yan et al. [41] aim to predict scores that are calibrated to the item labels, as their scores are both used for ranking *and* for downstream tasks. Though our proposed model does not require scores to be calibrated, we leverage their observation that popular pair- and listwise loss functions are invariant to the absolute scale of scores, which causes them to miss an important signal during training.

Finally, note that much of the recent LTR literature has been interested in *unbiased* LTR [20], which considers the bias in clicking behavior depending on the order in which items are shown to the user. We will not consider this bias in our proposed model to reduce its complexity, i.e. our proposed model is permutation-invariant with respect to the list ordering. A study on how the overall quality of the list (and thus also the listwide label) may depend on the list ordering is left to future work.

3 BACKGROUND

Our contributions are aimed at the ranking problem, which we formalize in Sec. 3.1. In Sec. 3.2, we briefly review how such problems are typically solved through Learning-to-Rank. A particular ranking model that is core to our proposed method is the listwise Transformer, which we discuss in Sec. 3.3.

3.1 Ranking

We use $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ to denote the real-valued, d_x-dimensional feature vector corresponding with the occurrence item $i \in I$ in a selection list I. For example in search engines, these features may be solely based on the item itself or also dependant on a search query and other contextual information. Based on the response to the selection, feedback values $y_i \in \mathcal{Y} = \{0, ..., y_{\max}\}$ are gathered for each item $i \in I$. The feedback values are assumed to be ordinal, where higher values indicate that the user had a stronger preference for the item. In this work, we focus on *implicit* feedback, e.g. in the form of clicks and purchases, which is inferred from a user's overall interaction with the selection. This is opposed to *explicit* feedback, where the user directly assign scores to items. A key difference is that implicit labels often make no distinction between 'negative' and 'missing' feedback. For example, $y_i = 0$ may indicate that item i was bad, or that item i was ignored and thus never explicitly considered.

Let $\boldsymbol{X}_I = (\boldsymbol{x}_i^\top)_{i \in I}$ denote the matrix of all feature vectors in list I and $\boldsymbol{y}_I = (y_i)_{i \in I}$ the vector of all its labels. The ranking dataset $\mathcal{D} = \{(\boldsymbol{X}_I, \boldsymbol{y}_I) \mid I \in \mathcal{I}\}$ then gathers all lists in the superset $\mathcal{I}$. Note that in realistic search engine settings, it is likely that the same item shows up in many lists. As such, when we refer to item i, we mean the occurrence of an item in list I where it had features $\boldsymbol{x}_i$ and label y_i. Moreover, our experiments allow for the case where the same selection with the same features shows up in lists A and B, though the collected user feedback may be different: $\boldsymbol{X}_A = \boldsymbol{X}_B \implies \boldsymbol{y}_A = \boldsymbol{y}_B$.

3.2 Learning-to-Rank

The goal of Learning-to-Rank (LTR) [25] is to produce rankings that lead to a high utility. Typically, this is done using a scoring function $\boldsymbol{s} : \mathbb{R}^{|I| \times d_x} \to \mathbb{R}^{|I|}$ that produces a vector of scores $\boldsymbol{s}(\boldsymbol{X}_I)$ for list I with features $\boldsymbol{X}_I$. The individual items i are then ranked in the descending order of $\boldsymbol{s}(\boldsymbol{X}_I)_i$ scores. Traditional scoring functions are *pointwise*, i.e. each score $\boldsymbol{s}(\boldsymbol{X}_I)_i = s(\boldsymbol{x}_i)$ is computed individually per feature vector $\boldsymbol{x}_i$.

In *Neural* LTR approaches, $\boldsymbol{s}$ is a parameterized neural network for which a loss function is optimized that compares scores $\boldsymbol{s}(\boldsymbol{X}_I)$ to labels $\boldsymbol{y}_I$. This comparison is either made *pointwise* [23] (i.e. $\boldsymbol{s}(\boldsymbol{X}_I)_i$ with y_i), *pairwise* [8, 9] (i.e. $(\boldsymbol{s}(\boldsymbol{X}_I)_i, \boldsymbol{s}(\boldsymbol{X}_I)_j)$ with (y_i, y_j) for $i, j \in I$), or *listwise* (i.e. $\boldsymbol{s}(\boldsymbol{X}_I)$ with $\boldsymbol{y}_I$). In particular, we use the *ListNet* [10] or *Softmax* loss. The Softmax loss L_y is defined as

$$L_y(\boldsymbol{s}(\boldsymbol{X}_I), \boldsymbol{y}_I) = -\boldsymbol{y}_I^\top \log \boldsymbol{\sigma}(\boldsymbol{s}(\boldsymbol{X}_I)) \tag{1}$$

which is simply the dot-product of the label vector $\boldsymbol{y}_I$ and the elementwise log of the softmax function $\boldsymbol{\sigma}(\mathbf{a})_i = \frac{\exp(\mathbf{a}_i)}{\sum_{j \in I} \exp(\mathbf{a}_j)}$ for $i \in I$, applied to the scores $\boldsymbol{s}(\boldsymbol{X}_I)$.

If the ranking is done according to a specific stochastic model (i.e. ranking permutations are randomly sampled from a distribution that is influenced by the scores $s(\boldsymbol{X})$) and if the label domain is binary (i.e. $\forall i : y_i \in \{0, 1\}$), then Softmax loss indicates the likelihood of item i to be ranked first [6, 10]. Though we do not constrain the label domain to be binary, we have found the Softmax loss to show strong empirical results while being simple and efficient. This is aligned with recent findings in similar work [34, 37].

3.3 The Listwise Transformer

Though scoring functions $\boldsymbol{s}$ usually compute pointwise scores, i.e. independently per item, it may be beneficial to make the score of an item i dependant on other items $\{j \mid j \in I \wedge j \neq i\}$ in the selection I. E.g. in e-commerce, if a product i is much cheaper than the other products that will be shown, then price-related features in $\boldsymbol{x}_i$ may be more important than when all products have the same price.

A *listwise* scoring function $\boldsymbol{s}$ can be constructed by having a computation step that transforms the feature matrix $\boldsymbol{X}_I$ in each list I such that information is shared between items $i \in I$. This kind of transformation can be learned end-to-end by applying an attention mechanism over $\boldsymbol{X}_I$, e.g. through the use of the *Transformer* [38] architecture that has seen tremendous success in Natural Language Processing (NLP) and has recently also been applied to ranking problems [24, 30, 34, 37].

The core component of the Transformer architecture is the *Self-Attention* (SA) layer. Here, each feature vector $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ in the feature matrix $\boldsymbol{X}_I \in \mathbb{R}^{|I| \times d_x}$ is independently sent through a shared *query* function $\boldsymbol{q} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_{\text{att}}}$, *key* function $\boldsymbol{k} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_{\text{att}}}$, and *value* function $\boldsymbol{v} : \mathbb{R}^{d_{\text{val}}} \to \mathbb{R}^{d_{\text{val}}}$, with $d_{\text{att}} = d_{\text{val}} = d_x$ in our implementation. For each input vector $\boldsymbol{x}_i$, the output value of the SA layer is then the listwise sum of value matrix $\boldsymbol{v}(\boldsymbol{X}_I) = (\boldsymbol{v}(\boldsymbol{x}_i)^\top)_{i \in I}$, weighted by the softmax $\boldsymbol{\sigma}$ of the dot-product between its query vector $\boldsymbol{q}(\boldsymbol{x}_i)$ and the matrix of key vectors $\boldsymbol{k}(\boldsymbol{X}_I) = (\boldsymbol{k}(\boldsymbol{x}_i)^\top)_{i \in I}$. The SA operation is thus defined as

$$\text{SA}(\boldsymbol{X}_I)_i = \boldsymbol{v}(\boldsymbol{X}_I)^\top \boldsymbol{\sigma}\left(\frac{\boldsymbol{k}(\boldsymbol{X}_I)\boldsymbol{q}(\boldsymbol{x}_i)}{\sqrt{d_{\text{att}}}}\right). \tag{2}$$

A typical Transformer encoder layer is made up of an SA block followed by a shared *Feed-Forward* (FF) network that processes each vector in the sequence independently. Furthermore, the SA and FF blocks often contain residual, Dropout and LayerNorm layers. For a full discussion, we refer to the original work [38] and comparative

technical overviews [33]. Details on our exact implementation can be found in Appendix B.

We define $\mathrm{TF} : \mathbb{R}^{|I| \times d_x} \to \mathbb{R}^{|I| \times d_x}$ as the full Transformer encoder that, given feature matrix $\boldsymbol{X}_I$ as input, returns the list-aware feature matrix $\boldsymbol{Z}_I = \mathrm{TF}(\boldsymbol{X}_I)$. The transformed feature vectors $\boldsymbol{z}_i$ correspond with items i, but were computed knowing the context of other items in list I. To finally construct the *listwise* scoring function $\boldsymbol{s}$, we therefore apply a pointwise, shared scoring head $h_s : \mathbb{R}^{d_x} \to \mathbb{R}$ to each z_i:

$$\boldsymbol{s}(\boldsymbol{X}_I)_i = h_s(\mathrm{TF}(\boldsymbol{X}_I)_i) \tag{3}$$

where h_s is a learnable, feed-forward network.

4 LISTWIDE LABELS

In contrast to the *listwise* LTR defined in Sec. 3.2, we now discuss *listwide* labels. These quantify the overall quality of the list.

A key use case of listwide labels is the ability to learn from lists where all labels are zero, which we motivate in Sec. 4.1. We then formulate a precise definition of listwide labels in Sec. 4.2 and a strategy to learn from them in Sec. 4.3.

4.1 Lists without Relevancy Labels

Performing listwise LTR can deliver strong results because it jointly evaluates the scoring function $\boldsymbol{s}(\boldsymbol{X}_I)$ over all items $i \in I$ with respect to the user feedback labels $\boldsymbol{y}_I$. Yet, listwise (and pairwise) LTR loss functions only consider the values of scores and labels at separate, relative scales per list. For example, if a label y_i is low, then it will only be considered low in that list by pair- and listwise loss functions if there are other labels $y_j > y_i$ in the same list. Many such loss functions are even completely invariant to global translations of the scores vector $\boldsymbol{s}(\boldsymbol{X}_I)$, which means scores are not necessarily calibrated to the labels they try to align with [41].

Hence, constant lists with value c, i.e. $\forall i \in I : y_i = c$, provide no useful information to a pair- or listwise loss function [39]. Indeed, for constant lists with $c \neq 0$ the L_y loss in Eq. 1 will push all scores $\boldsymbol{s}(\boldsymbol{X}_I)$ to simply be constant, regardless of whether c is high or low. In the particular case of lists without relevancy labels, i.e. $\forall i \in I : y_i = 0$, we even have the Softmax loss $L_y(\cdot, \boldsymbol{0}) = 0$. Though pointwise loss functions *do* consider the absolute scale of individual labels y_i, they are otherwise empirically inferior to the best pair- and listwise loss functions in LTR benchmarks [36, 37].

In practice, lists without relevancy labels therefore tend to be ignored during optimization. This either happens *implicitly* because pair- or listwise loss functions are typically zero or constant with respect to the scores, or *explicitly* because they are filtered out during preprocessing [6, 34, 39].

However, lists without relevancy labels are actually common in real-world implicit feedback, since users often do not interact with lists. We argue such lists should not be ignored, as this could be an important signal about the quality of the ranking. For example in the e-commerce setting, $\boldsymbol{y}_I = \boldsymbol{0}$ could mean that none of the products in I where desirable, so the user may have tried a different query or store. Indeed, internal data at Amazon Search shows that many queries do not result in any user feedback. To better capture this signal, we propose that the score function $\boldsymbol{s}$ should learn that the products in the selection all had the same (poor) quality.

A straightforward solution could therefore be to exclusively use pointwise LTR loss functions for those lists. Yet, this approach still misses that the constant quality of label vectors such as $\boldsymbol{y}_I = \boldsymbol{0}$ may, in part, be influenced by the *overall* quality of I. As concluded in the user study by Joachims et al. [19]: "clickthrough data does not convey *absolute* relevance judgments, but partial *relative* relevance judgments for the links the user evaluated".

Going one step further, we suggest that it may not suffice to treat lists with $\boldsymbol{y}_I = \boldsymbol{0}$ as a combination of items that *individually* had the same poor quality, but rather as a set of items that *together* received no feedback. Hence, we propose to learn from this signal at the list level, as opposed to decomposing it over the item level.

4.2 Deriving Listwide Labels

We introduce the notation $t_I \in \mathcal{T}$ to refer to the overall quality of the list I, i.e. the *listwide* label. To generate informative listwide labels, in particular for $\boldsymbol{y}_I = \boldsymbol{0}$, we derive t_L from the item labels. In our case, we set t_I as the *maximal* implicit feedback label:

$$t_I = \max_{i \in I} y_i \tag{4}$$

where it is practical that $t_I = 0 \iff \boldsymbol{y}_I = \boldsymbol{0}$ with item labels $y_i \in \mathcal{Y} = \{0, ..., y_{\max}\}$. Also, note that the domain of the item labels is inherited by the listwide labels $t_I \in \mathcal{T} = \mathcal{Y}$.

This formulation for t_I is motivated by the observation that in most ranking contexts, users are typically looking for a single 'thing'. Overall user satisfaction, which is a long-term goal in web and e-commerce search, can thus be approximated by observing whether the user's overall goal was reached. For example, that goal might be "finding a relevant document" during a web search or "buying a suitable product" in e-commerce.

4.3 Learning from Listwide Labels

To learn from listwide labels, we define $d : \mathbb{R}^{|I| \times d_x} \to \mathcal{T}$ as the function that estimates the listwide quality t_I. For a parameterized d, we can then optimize the listwide loss function $L_t(d(\boldsymbol{X}_I), t_I)$ for list I. As the values $d(\boldsymbol{X}_I)$ and t_I are singletons with a domain inherited from the item labels (through Eq. 4), any pointwise loss function for the item labels in LTR is also applicable for the definition of L_t.

Considering that $\mathcal{T} = \mathcal{Y} = \{0, ..., y_{\max}\}$ is ordinal, we make use of the *Ordinal* loss function [29] as applied to ranking [34]. To this end, we use $\boldsymbol{\omega} : \mathcal{T} \to \{0, 1\}^{y_{\max}}$ as the function that converts the listwide labels t_I into their ordinal, one-hot encoding $\boldsymbol{\omega}(t_I)$, i.e.

$$\boldsymbol{\omega}(t_I)_k = \begin{cases} 1 & \text{if } t_I \geq k \\ 0 & \text{else} \end{cases} \tag{5}$$

with $k \in \{1, ..., y_{\max}\}$.

For example with $\mathcal{T} = \{0, 1, 2\}$, we would have vectors $\boldsymbol{\omega}(0) = (0, 0)$, $\boldsymbol{\omega}(1) = (1, 0)$ and $\boldsymbol{\omega}(2) = (1, 1)$.

Furthermore, we now change the listwide predictor such that it is also vector-valued, i.e. $\boldsymbol{d} : \mathbb{R}^{|I| \times d_x} \to [0, 1]^{y_{\max}}$ The *Ordinal* loss function L_t is then defined as

$$L_t(\boldsymbol{d}(\boldsymbol{X}_I), t_I) = \sum_{k=1}^{y_{\max}} \mathrm{BCE}\left(\boldsymbol{d}(\boldsymbol{X}_I)_k, \boldsymbol{\omega}(t_I)_k\right) \tag{6}$$

which is the sum of the Binary Cross-Entropy (BCE) losses for each non-zero $k \in \mathcal{T}$. The ordinality of L_t results from the fact that if

$\boldsymbol{d}(\boldsymbol{X}_I)_k$ is high, then the estimated likelihood of $\boldsymbol{d}(\boldsymbol{X}_I)_l$ for $l < k$ should also be high. For example in e-commerce, if $t_I = 1$ implies that the user clicked something and $t_I = 2$ implies they purchased something, then $\boldsymbol{d}$ should only predict that a purchase was made if it is also predicting a click.

Finally, note that the benefit of listwide assessments not only results from being able to learn from lists with constant feedback (e.g. $\boldsymbol{y}_I = \boldsymbol{0}$). It also has the practical advantage that it allows us to make a quality assessment about selections that is easy to aggregate over large sets of selections. For example in e-commerce, predicting t_I allows us to judge whether new selections I will be successful or not, before they are shown to customers.

5 RANKFORMER

Learning from the list*wide* labels discussed in Sec. 4 allows us to fill in the gaps where list*wise* LTR, as discussed in Sec. 3.2, is not possible (e.g. when $\boldsymbol{y}_I = \boldsymbol{0}$). We therefore propose the *RankFormer* architecture, illustrated in Fig. 1, which is an extension of the listwise Transformer architecture discussed in Sec. 3.3 that performs both listwise ranking and listwide quality predictions. By explicitly assessing the overall quality of a selection I, the individual scores for items $i \in I$ can be computed in the context of that overall quality.

To make predictions about the entire lists, we follow the standard NLP approach by adding a shared `[CLS]` token to each list. Since all tokens in listwise Transformers are expected to be feature vectors, we also define the learnable parameter vector $\boldsymbol{x}_{\texttt{[CLS]}}$ that is the same for every list. The items in the appended list $I^* = I \cup \texttt{[CLS]}$, with feature matrix $\boldsymbol{X}_{I^*}$, are then provided as input to the Transformer operation TF as normal. From the transformed features $\boldsymbol{Z}_{I^*} = \mathrm{TF}(\boldsymbol{X}_{I^*})$ returned as output, we extract the $\boldsymbol{z}_{\texttt{[CLS]}}$ vector that should reflect the information in $\boldsymbol{X}_I$ that was particularly relevant to judge the overall quality of I.

The actual listwide prediction $\boldsymbol{d}$ is made by applying a separate feed-forward network block $\boldsymbol{h}_d : \mathbb{R}^{d_x} \to [0,1]^{y_{\max}}$ to $\boldsymbol{z}_{\texttt{[CLS]}}$:

$$\boldsymbol{d}(\boldsymbol{X}_I) = \boldsymbol{h}_d(\mathrm{TF}(\boldsymbol{X}_{I^*})_{\texttt{[CLS]}}) \tag{7}$$

where we recall that $\boldsymbol{h}_d$ is vector-valued such that the Ordinal loss can be computed.

The listwise scores $\boldsymbol{s}(\boldsymbol{X}_I)$ could be computed as normal. However, we can make sure that the individual scores assigned to transformed feature vectors $\boldsymbol{z}_i$ for $i \in I$ are benefitting from the overall list quality information $\boldsymbol{z}_{\texttt{[CLS]}}$ by concatenating the latter with each of the $\boldsymbol{z}_i$ vectors. The listwise LTR scores $\boldsymbol{s}(\boldsymbol{X}_I)$ are thus computed as

$$\forall i \in I : \boldsymbol{s}(\boldsymbol{X}_I)_i = h_s(\mathrm{TF}(\boldsymbol{X}_{I^*})_i \parallel \mathrm{TF}(\boldsymbol{X}_{I^*})_{\texttt{[CLS]}}) \tag{8}$$

with $\parallel$ the concatenation operator and with the shared scoring head now $h_s : \mathbb{R}^{2d_x} \to \mathbb{R}$.

The RankFormer, with its learnable parameters in $\boldsymbol{x}_{\texttt{[CLS]}}$, TF, h_s and $\boldsymbol{h}_d$, thus contains all the necessary components to perform listwise LTR while making listwide predictions. A straightforward way to optimize these parameters is through the minimization of the sum of the listwise loss L_y and the listwide loss L_t:

$$\min_{\boldsymbol{x}_{\texttt{[CLS]}},\mathrm{TF},h_s,\boldsymbol{h}_d} \sum_{I \in \mathcal{I}} L_y(s(\boldsymbol{X}_I), \boldsymbol{y}_I) + \alpha L_t(\boldsymbol{d}(\boldsymbol{X}_I), t_I). \tag{9}$$

where the hyperparameter α controls the strength of L_t.

Ending our discussion of the RankFormer, we note that it does not use the positional embeddings that are popular in the application of Transformers in NLP [38]. The listwise LTR scores are thus permutation-equivariant [30] with respect to $\boldsymbol{X}$, whereas the listwide quality predictions are permutation-invariant. Here, the use of positional embeddings is not necessary because we assume the selection I to be unordered with no prior ranking. We refer to related work [34] for a study on the impact of this assumption.

6 EXPERIMENTS ON PUBLIC DATASETS

We performed experiments on three public datasets: MSLR-WEB30K [35], Istella-S [27] and Set 1 of the Yahoo [11] dataset. All consist of web search data, where humans were asked to assign an explicit relevance label to each URL in the selection.

However, our focus is on learning from implicit labels, which are far more abundant in realistic applications. As is done in Unbiased LTR literature [20], we therefore simulate implicit labels in a manner that aligns with realistic user behavior seen in industry. This is discussed in Sec. 6.1.

We further detail our evaluation setup in Sec. 6.2 and finally report our results in Sec. 6.3.

6.1 Simulating Implicit Feedback

Let $r_i \in \{0, ..., r_{\max}\}$ denote the *explicit* relevance label assigned to item $i \in I$. It so happens that all our datasets have $r_{\max} = 4$. For example in Yahoo, gradings r_i correspond with

$$\{\texttt{bad}, \texttt{fair}, \texttt{good}, \texttt{excellent}, \texttt{perfect}\}.$$

Recall that *implicit* relevance labels $y_i \in \{0, ..., y_{\max}\}$ are inferred from user behavior. In what follows, we explicitly set $y_{\max} = 2$ and interpret the domain of y_i as

$$\{\texttt{seen}, \texttt{click}, \texttt{conversion}\}.$$

Assumption 1. *Implicit labels y_i are positively correlated with explicit labels r_i.*

To put Assumption 1 into practice, we use the function ρ, which maps the relevance grades r_i to a probability of relevance [2, 12]:

$$\rho(r_i) = \frac{2^{r_i} - 1}{2^{r_{\max}} - 1}. \tag{10}$$

In an effort to align the exact relationship between r_i and y_i with experiences from industry, we perform our simulation in three stages: *Selection* where subsets of lists are sampled, *Intent* where an overall list assessment is made, and *Interaction* where the actual implicit label y_i is generated.

6.1.1 Selection. A common effect in LTR applications is *selection bias* [18], i.e. though a selection $J \in \mathcal{I}$ was gathered, users often only examine a subset of the selection $I \subset J$ with a maximal length N_s.

Assumption 2. *The actual selections I seen by users are small.*

For example, Assumption 2 holds when only N_s items are shown on a single webpage and users do not navigate to later pages. A practical way to generate such lists is to randomly sample up to N_s items without replacement.

However, this step leads to many sampled lists I that exclusively contain items with low r_i values. In practice, it is indeed realistic that

many selections have an overall low relevance, e.g. because users gave an uninformative query. Yet, it may be too difficult to achieve meaningful validation and test set performance on public datasets because many items in the original training set will be removed. For each of the original lists J, we therefore perform bootstrapping by sampling N_b selections I. This also mimics a scenario where not all users see exactly the same list for the same query

In our experiments, we set $N_s = 16$ and $N_b = 10$.

6.1.2 Intent. Our aim is to explicitly simulate the *intent* of the user in interacting with the list, e.g. whether the user wants to only view the selection, or whether they want to invest more time and also click and buy things [28]. To this end, we take inspiration from the user study by Liu et al. [26] where it is observed that users first skim many of the items in the list, before giving any of them serious consideration. We assume that the overall relevancy $\boldsymbol{r}_I = (r_i)_{i \in I}$ observed during this skimming step affects the user intent.

Assumption 3. *Users intend stronger interactions with lists I that have an overall higher relevance $\boldsymbol{r}_I$.*

Let T_I denote the intent of the user for list I, which we consider as the maximal action that the user is willing take. Thus $T_I \in \mathcal{Y}$, since we measure user behavior as implicit feedback. To model Assumption 3, we echo the definition of listwide labels in Eq. 4 and consider the overall list quality as the highest explicit label:

$$P(T_I \mid \boldsymbol{r}_I) = \begin{cases} 0 & 1 - \rho(\max_{i \in I} r_i) \\ 1 & (1-\kappa)\rho(\max_{i \in I} r_i) \\ 2 & \kappa\rho(\max_{i \in I} r_i) \end{cases} \tag{11}$$

with $\kappa \in [0, 1]$ a hyperparameter that controls how often clicks result in conversions. It is common that $\kappa \ll 0.5$ in e-commerce settings, since users mostly only want to browse [28].

The pragmatic benefit of Eq. 11 is that it allows us to simulate all-zero lists, as they are guaranteed for intent $T_I = 0$. Some simulations of implicit feedback in prior work, such as by Jagerman et al. [18], were also capable of generating all-zero lists. However, they do not do so by simulating the overall intent of users and do not assume a dependency on the overall quality of the list.

In our experiments, we set $\kappa = 0.1$.

6.1.3 Interaction. Finally, given the intent T_I for the observed list I, we sample individual implicit labels y_i.

Assumption 4. *Higher implicit labels y_i are less noisy.*

We also motivate Assumption 4 from the e-commerce setting, where conversions are more robust than clicks [21]. For example, this is because higher y_i labels involve a higher (financial) cost, so more thought is put into the action.

For $y_i \in \{0, 1, 2\}$, we first decide conversions ($y_i = 2$) and then clicks ($y_i = 1$) for all items that did not have a conversion, i.e. $y_i < 2$. Given the overall list intent T_I, we sample these decisions with likelihoods

$$\begin{aligned} P(y_i = 2 \mid r_i, T_I = 2) &= \rho(r_i) \\ P(y_i = 1 \mid r_i, T_I \geq 1, y_i < 2) &= \epsilon + (1-\epsilon)\rho(r_i). \end{aligned} \tag{12}$$

Here, $\epsilon \in [0, 1]$ is a parameter that represents the noise in click behavior [2]. Following Assumption 4, no explicit noise is simulated for $y_i = 2$. Moreover, following our definition of the list intent T_I, we clarify that $\forall k : P(y_i = k \mid T_I < k) = 0$.

In experiments, we set $\epsilon = 0.1$ as was previously done in work on unbiased LTR [2]. However, in contrast to unbiased LTR settings [2, 20], we do not assume that the likelihood of clicks is dependant on the item's position in the shown list.

6.2 Evaluation Setup

Table 1: Statistics of the public datasets. The selection stage in the simulation keeps at most $N_s = 16$ items per list. Each query is sampled $N_b = 10$ times per original list. For WEB30K, the statistics are computed for Fold 1.

		WEB30K	ISTELLA	YAHOO
# features		136	220	700
# original lists	train	18,919	19,245	19,944
	valid	6,306	7,211	2,994
	test	6,306	6,562	6,983
avg. length	original	119.6	103.2	23.7
	sampled	15.7	16.0	12.7

Each configuration is run for five random seeds. For the WEB30K dataset, each seed indexes a different fold out of five different pre-defined dataset splits. For ISTELLA and YAHOO there is only a single pre-defined fold, so we re-use the same train/validation/test split for each seed. The implicit labels generated per random seed and dataset are the same for each method evaluation. Overall statistics on the datasets are given in Tab. 1. The distributions of these simulated labels are given in Fig. 2. They largely correspond with statistics on internal datasets at Amazon Search.

For WEB30K and ISTELLA, we use a quantile transform, trained on each training data split, with a normal output distribution. The YAHOO data already provides quantile features, so we just apply a standard transformation to also make them normally distributed.

6.2.1 Methods. In Sec.5, we proposed the RankFormer architecture. To validate the benefit of the listwide loss, we evaluate the RankFormer for $\alpha \in \{0, 0.25, 1\}$, where $\alpha = 0$ implies that no listwide loss is used. The architecture then mostly reverts to the listwise Transformer discussed in Sec. 3.3, which has been evaluated using explicit labels in prior work [30, 34, 37].

We evaluate the benefit of listwise attention by comparing these methods to a simple *Multi-Layer Perceptron* (MLP) as a baseline. Moreover, as it is known that neural LTR methods often struggle to compete with tree-based models [37], we train a *Gradient-Boosted Decision Tree* (GBDT) [13] using the LightGBM [22] framework with the LambdaRank [7] objective. We noticed a substantial benefit from using GBDTs with more trees than previous benchmarks. Therefore, we report results with both 1,000 (as in [37]) and 10,000 trees.

All methods optimize an LTR loss (Softmax for the MLP and RankFormer, LambdaRank [7] for the GBDT) over the implicit labels for all training set lists I where $t_I > 0$ (i.e. $\exists i \in I : y_i > 0$). Only the RankFormer with $\alpha > 0$ also trains on lists with $t_I = 0$. A hyperparameter search was done for the most important hyperparameters of each method, with the validation NDCG_y as the objective. We refer to Appendix D for further details.

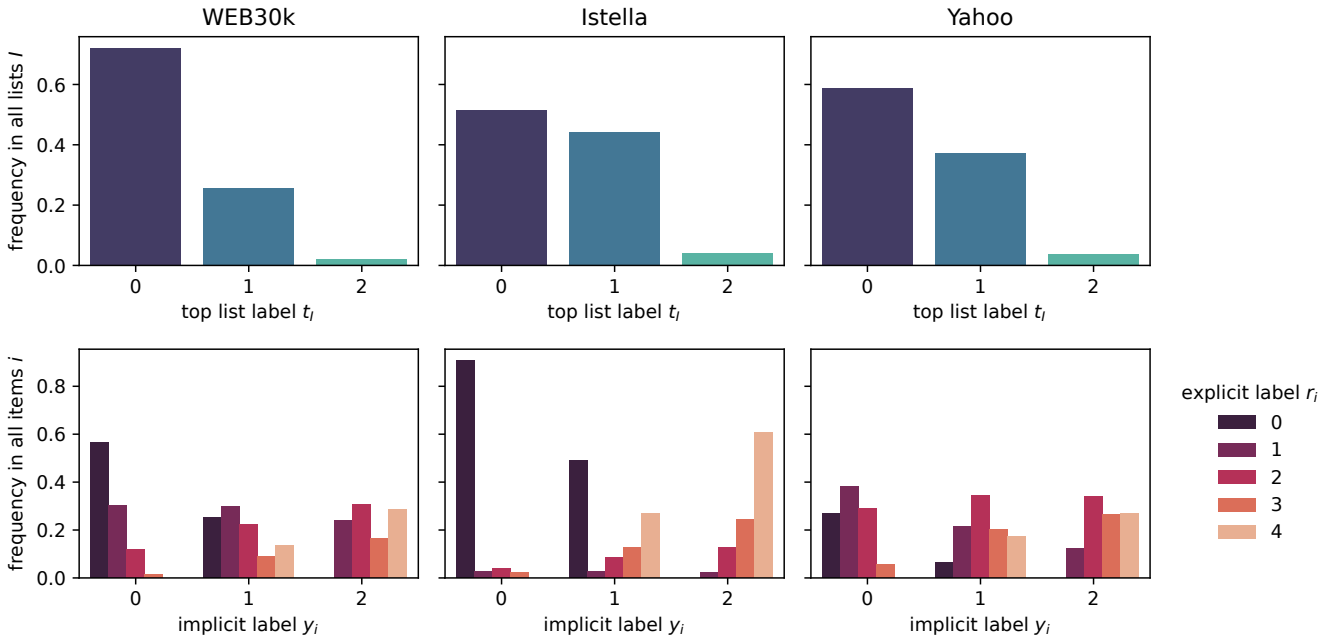

Figure 2: The simulated distribution of the lists' top implicit labels $t_I = \max_{i \in I} y_i$ and the distributions of the individual explicit labels r_i per simulated implicit label y_i.

Table 2: NDCG@10 on the WEB30K, ISTELLA and YAHOO datasets. NDCG_y refers to NDCG as measured on simulated, *implicit* labels. NDCG_r is computed over the original, *explicit* labels. Each result is the mean out of five runs, ± the standard error. The GBDT with 10^4 trees is superior in almost all cases, so we show the *two* configurations with the best results in bold.

		WEB30K		ISTELLA		YAHOO	
		NDCG_y	NDCG_r	NDCG_y	NDCG_r	NDCG_y	NDCG_r
GBDT	# trees = 10^3	51.17 ± 0.17	63.01 ± 0.09	67.50 ± 0.05	86.99 ± 0.02	**66.30 ± 0.02**	79.35 ± 0.03
	# trees = 10^4	**51.53 ± 0.14**	63.52 ± 0.09	**68.42 ± 0.04**	**88.75 ± 0.01**	**66.82 ± 0.04**	**80.71 ± 0.02**
MLP		50.74 ± 0.10	63.13 ± 0.07	67.63 ± 0.06	87.38 ± 0.04	65.94 ± 0.08	**79.59 ± 0.03**
RankFormer	$\alpha = 0$	51.19 ± 0.08	63.78 ± 0.10	67.83 ± 0.08	87.75 ± 0.03	65.82 ± 0.03	79.27 ± 0.15
	$\alpha = 0.25$	**51.32 ± 0.12**	**63.97 ± 0.13**	**67.95 ± 0.07**	**87.88 ± 0.04**	65.95 ± 0.09	79.51 ± 0.11
	$\alpha = 1$	51.19 ± 0.09	**63.84 ± 0.08**	67.85 ± 0.06	87.82 ± 0.02	65.23 ± 0.10	78.22 ± 0.27

6.2.2 Metrics. The test set predictions in each experiment run are evaluated with the NDCG@10 metric, multiplied by 100. In our results, we report both NDCG_y, i.e. the NDCG with respect to the simulated *implicit* labels y_i (see Sec. 6.1), and the NDCG_r, i.e. the NDCG with respect to the original *explicit* labels r_i. Lists with constant label vectors $\boldsymbol{y}_I$ or $\boldsymbol{r}_I$ respectively are not taken into account for the mean aggregation of the implicit or explicit NDCG.

Importantly, these metrics are only computed on the *sampled* lists according to Sec. 6.1.1. This means that the NDCG computed on explicit labels is not comparable to other benchmarks, where the full selection (with better and worse items) is used instead. Note that the scores would anyhow not be comparable, since all methods are trained on simulated, implicit labels. To anyhow validate our experiment pipeline, we also report experiment results on the original WEB30K dataset in Appendix A, where the same hyperparameters were used as for the simulated experiments.

6.3 Results

We report our results in Tab. 2. They largely confirm the common finding that strong GBDTs are superior to neural methods (e.g. the MLP and RankFormer) on tabular data [16]. Indeed, the strongest neural ranking methods already struggle to outperform a GBDT with 1,000 trees in recent benchmarks [30, 37]. We now allow GBDTs an even higher complexity (i.e. 10,000 trees) and find that the advantage of GBDTs is even more distinctive: a GBDT model scores best on two (ISTELLA and YAHOO) out of three datasets.

Yet despite the fact that GBDTs remain state-of-the-art on purely tabular data, neural methods may still hold more potential, e.g. because they integrate well with other data formats such as raw text and images. When only considering the neural methods, we see that the RankFormer with $\alpha = 0.25$ clearly benefits from learning with listwide labels. This confirms our hypothesis that there is value

in learning from lists with no labels. However, the choice of α is important, as a larger focus on the listwide loss leads to a smaller focus on the listwise LTR objective. This can ultimately lead to worse NDCG scores (e.g. for $\alpha = 1$).

Finally, we note that there is no advantage for the RankFormer over the MLP baseline on the Yahoo dataset. The fact that Transformer-based methods appear to perform poorly on Yahoo is also aligned with previous benchmarks [30, 37].

Overall, we conclude that the RankFormer is superior to the state-of-the-art in neural LTR when listwide labels are provided. This advantage is seen both in the NDCG as measured on the simulated, implicit labels and the original, explicit labels.

7 EXPERIMENTS ON AMAZON SEARCH

Much of the motivation for our proposed RankFormer method and our simulation of implicit labels has come from the application of LTR to e-commerce product search. In this section, we report results on Amazon Search data, which is collected from users interacting with Amazon websites. This results in implicit feedback, e.g. *clicks* and *purchases*.

We run both *offline* and *online* experiments.

7.1 Offline Experiments

Table 3: NDCG$_y$@10 on an e-commerce dataset at Amazon Search, computed over organic *implicit* labels. We report the overall NDCG$_y$ and the NDCG$_y$ specifically for queries that resulted in a purchase (i.e. $t_I = 2$).

		Amazon Search (offline)	
		NDCG$_y$	NDCG$_y(t_I = 2)$
GBDT	# trees = 10^3	35.11	50.00
	# trees = 10^4	37.07	53.70
MLP		36.54	53.98
RankFormer	$\alpha = 0$	37.05	54.26
	$\alpha = 0.25$	**37.33**	**55.20**
	$\alpha = 1$	37.18	54.36

The training set consists of 500k queries sampled from a month of customer interactions. Additionally, we gathered 139k validation set queries and 294k test set queries, both collected over one week. Between each set, there is a span of two weeks where no data is used to prevent leakage. In contrast to the public dataset experiments, we use organically collected implicit values without simulation. Our evaluation setup was similar to Sec. 6.2, though no explicit labels are available for evaluation.

We report the results in Tab. 3. Here, the RankFormer model clearly outperforms the baseline methods, especially on the NDCG$_y$ measured over queries that resulted in a purchase (i.e. $t_I = 2$). It sees a further benefit for $\alpha > 0$, indicating that the RankFormer is successful in learning from listwide labels. Since many queries have no individual product interactions ($\boldsymbol{y}_I = 0$), listwide labels can indeed be a useful signal.

Compared to experiments on public datasets in Tab. 2, it is noticeable that the neural methods outperform the most powerful GBDTs. In fact, a GBDT with 10,000 trees is far too complex to meet the latency constraints required for online inference. An important factor contributing to this superiority of neural methods is that a richer set of features is available in the Amazon Search data than in the public dataset. For example, neural models can directly learn from raw text and image features, or from dense feature vectors generated by upstream neural models.

7.2 Online Experiments

The superiority of the RankFormer method in offline experiments may not translate to an advantage in practice. This can only be fully validated through online experiments. Yet, though the RankFormer benefits from the high parallelizability of the Transformer, its implementation demands significant changes to our internal pipeline, in particular because it performs listwise scoring as opposed to the pointwise scoring done by the GBDT and MLP. Therefore, we employ *knowledge distillation* [15] by training a GBDT with 2,000 trees on the scores given by either the MLP or RankFormer (with $\alpha = 0.25$) that performed best in Tab. 3 as teacher models.

These two models (i.e. the GBDT with the MLP teacher and the GBDT with the RankFormer teacher) were then compared in an interleaving experiment as described in [5]. Interleaving has a much greater statistical power than A/B tests, because the same customer is presented with a mix of the rankings of both models. The comparison between methods is quantified as the *credit* lift, i.e. the extent to which the user's actions are attributed to either model. It was empirically shown in [5] that this interleaving credit is strongly correlated with corresponding A/B metrics.

Our interleaving experiment ran for two weeks. Overall, the GBDT with the RankFormer teacher was credited with 13.7% lift in revenue attributed to product search. The actual gain that would be seen in an A/B test is likely far smaller, yet the experiment shows that the GBDT with the RankFormer teacher is superior with $\boldsymbol{p = 0.02}$. Thus, we conclude that the listwise ranking performed by the RankFormer can be successfully translated to practical results, even when only distilling its listwise ranking scores to a pointwise GBDT scoring function. Though pointwise models lack the full list context, we hypothesize that powerful listwise rankers can help 'fill in' implicit labels that are missing or noisy in the original data. Moreover, listwise models may favor features that lead to more diverse rankings.

8 CONCLUSION

We formalized the problem of learning from listwide labels in the context of listwise LTR as a method to learn from both relative user feedback on individual items and the absolute user feedback on the overall list. Our proposed RankFormer architecture combines both objectives by jointly modelling both the overall quality of a ranking list and the individual utility of its items.

To conduct experiments on popular, public datasets, we simulated implicit feedback derived from the explicit feedback provided in those datasets. These experiments indicate that the RankFormer is indeed able to learn from listwide labels in a way that helps performance in LTR, thereby improving upon the state-of-the-art in neural ranking. However, in this tabular data setting, our results also show that strong GBDT models can be too powerful to be outclassed by neural methods.

On internal data at Amazon Search, where a richer set of features is available, the RankFormer achieves better performance offline than any of the other methods, including strong GBDT models. In an online experiment, we distill the RankFormer to a practically viable GBDT and observe that this advantage is maintained over a complex MLP distilled to a similar GBDT.

Our findings encourage future work in learning from listwide labels. The user's perspective is central to ranking problems, and this particular signal of user feedback appears ripe for further research.

ACKNOWLEDGMENTS

This research was supported by the TEN Search team at Amazon, the ERC under the EU's 7th Framework and H2020 Programmes (ERC Grant Agreement no. 615517 and 963924), the Flemish Government (AI Research Program), the BOF of Ghent University (PhD scholarship BOF20/DOC/144), and the FWO (project no. G0F9816N, 3G042220).

REFERENCES

[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 135–144. https://doi.org/10.1145/3209978.3209985

[2] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. 2018. Unbiased Learning to Rank with Unbiased Propensity Estimation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 385–394. https://doi.org/10.1145/3209978.3209986 arXiv:1804.05938 [cs]

[3] 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 (ICTIR '19)*. Association for Computing Machinery, New York, NY, USA, 85–92. https://doi.org/10.1145/3341981.3344218

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [cs, stat]

[5] Nan Bi, Pablo Castells, Daniel Gilbert, Slava Galperin, Patrick Tardif, and Sachin Ahuja. 2022. Debiased Balanced Interleaving at Amazon Search. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. ACM, Atlanta GA USA, 2913–2922. https://doi.org/10.1145/3511808.3557123

[6] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. ACM, Santa Clara CA USA, 75–78. https://doi.org/10.1145/3341981.3344221

[7] Christopher Burges. 2010. From Ranknet to Lambdarank to Lambdamart: An Overview. *Learning* 11 (Jan. 2010).

[8] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems*, Vol. 19. MIT Press.

[9] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/1102351.1102363

[10] 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 (ICML '07)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[11] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview. In *Proceedings of the Learning to Rank Challenge*. PMLR, 1–24.

[12] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected Reciprocal Rank for Graded Relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. Association for Computing Machinery, New York, NY, USA, 621–630. https://doi.org/10.1145/1645953.1646033

[13] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* 29, 5 (2001), 1189–1232. jstor:2699986

[14] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems* 6, 4 (Dec. 2016), 13:1–13:19. https://doi.org/10.1145/2843948

[15] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge Distillation: A Survey. *International Journal of Computer Vision* 129, 6 (June 2021), 1789–1819. https://doi.org/10.1007/s11263-021-01453-z

[16] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why Do Tree-Based Models Still Outperform Deep Learning on Tabular Data? arXiv:2207.08815 [cs, stat]

[17] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. 2020. A Deep Look into Neural Ranking Models for Information Retrieval. *Information Processing & Management* 57, 6 (Nov. 2020), 102067. https://doi.org/10.1016/j.ipm.2019.102067

[18] Rolf Jagerman, Harrie Oosterhuis, and Maarten de Rijke. 2019. To Model or to Intervene: A Comparison of Counterfactual and Online Learning to Rank from User Interactions. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 15–24. https://doi.org/10.1145/3331184.3331269 arXiv:1907.06412 [cs]

[19] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately Interpreting Clickthrough Data as Implicit Feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '05)*. Association for Computing Machinery, New York, NY, USA, 154–161. https://doi.org/10.1145/1076034.1076063

[20] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. Association for Computing Machinery, New York, NY, USA, 781–789. https://doi.org/10.1145/3018661.3018699

[21] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On Application of Learning to Rank for E-Commerce Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 475–484. https://doi.org/10.1145/3077136.3080838

[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

[23] Ping Li, Qiang Wu, and Christopher Burges. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems*, Vol. 20. Curran Associates, Inc.

[24] Yi Li, Jieming Zhu, Weiwen Liu, Liangcai Su, Guohao Cai, Qi Zhang, Ruiming Tang, Xi Xiao, and Xiuqiang He. 2022. PEAR: Personalized Re-ranking with Contextualized Transformer for Recommendation. In *Companion Proceedings of the Web Conference 2022*. ACM, Virtual Event, Lyon France, 62–66. https://doi.org/10.1145/3487553.3524208

[25] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (March 2009), 225–331. https://doi.org/10.1561/1500000016

[26] Yiqun Liu, Chao Wang, Ke Zhou, Jianyun Nie, Min Zhang, and Shaoping Ma. 2014. From Skimming to Reading: A Two-stage Examination Model for Web Search. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, Shanghai China, 849–858. https://doi.org/10.1145/2661829.2661907

[27] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. 2016. Post-Learning Optimization of Tree Ensembles for Efficient Ranking. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '16)*. Association for Computing Machinery, New York, NY, USA, 949–952. https://doi.org/10.1145/2911451.2914763

[28] Wendy W. Moe. 2003. Buying, Searching, or Browsing: Differentiating Between Online Shoppers Using In-Store Navigational Clickstream. *Journal of Consumer Psychology* 13, 1 (Jan. 2003), 29–39. https://doi.org/10.1207/S15327663JCP13-1&2_03

[29] Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. 2016. Ordinal Regression with Multiple Output CNN for Age Estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 4920–4928. https://doi.org/10.1109/CVPR.2016.532

[30] Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 499–508. https://doi.org/10.1145/3397271.3401104

[31] 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*. ACM, Virtual Event Norway, 145–148. https://doi.org/10.1145/3409256.3409819

[32] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, and Wenwu Ou. 2019. Personalized Re-ranking for Recommendation. arXiv:1904.06813 [cs]

[33] Mary Phuong and Marcus Hutter. 2022. Formal Algorithms for Transformers. https://doi.org/10.48550/arXiv.2207.09238 arXiv:2207.09238 [cs]

[34] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2021. Context-Aware Learning to Rank with Self-Attention. arXiv:2005.10084 [cs]

[35] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. https://doi.org/10.48550/arXiv.1306.2597 arXiv:1306.2597 [cs]

[36] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Information Retrieval* 13, 4 (Aug. 2010), 346–374. https://doi.org/10.1007/s10791-009-9123-y

[37] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. ARE NEURAL RANKERS STILL OUTPERFORMED BY GRADIENT BOOSTED DECISION TREES? (2021), 16.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

[39] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '16)*. Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/2911451.2911537

[40] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning Clicks into Purchases: Revenue Optimization for Product Search in E-Commerce. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, Ann Arbor MI USA, 365–374. https://doi.org/10.1145/3209978.3209993

[41] Le Yan, Zhen Qin, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2022. Scale Calibration of Deep Ranking Models. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*. Association for Computing Machinery, New York, NY, USA, 4300–4309. https://doi.org/10.1145/3534678.3539072

[42] Honglei Zhuang, Zhen Qin, Xuanhui Wang, Michael Bendersky, Xinyu Qian, Po Hu, and Dan Chary Chen. 2021. Cross-Positional Attention for Debiasing Clicks. In *Proceedings of the Web Conference 2021*. ACM, Ljubljana Slovenia, 788–797. https://doi.org/10.1145/3442381.3450098

A RESULTS ON THE ORIGINAL WEB30K

Table 4: NDCG@10 on the WEB30K, computed over the original, *explicit* labels and *without subsampled lists*. Each result is the mean out of five runs, ± the standard error.

		WEB30K NDCG_r
GBDT	# trees = 10^3	47.42 ± 0.12
	# trees = 10^4	51.05 ± 0.10
MLP		49.30 ± 0.10
RankFormer	$\alpha = 0$	**52.24 ± 0.08**
	$\alpha = 0.25$	49.75 ± 0.16

Our results on the public datasets reported in Tab. 2 were on the subsampled lists. This subsampling step was necessary to simulate realistic implicit feedback. To make these models more comparable to related work, we therefore run all our methods again on the original WEB30K without any of the simulation steps described in Sec. 6.1, yet with the same hyperparameters (as in Appendix D). These results are reported in Tab. 4.

In contrast to our experiments on simulated data, we now observe that the RankFormer with $\alpha = 0.25$ achieves far worse results than with $\alpha = 0$. The listwide loss may be less informative here, because the explicit labels were assigned independently per item by human annotators. The listwide quality effect is then far less important, which means smaller α values should be used.

B ARCHITECTURE DETAILS OF THE LISTWISE TRANSFORMER

The Transformer (TF) architecture that was used as a basis for the RankFormer, was implemented using the TransformerEncoderLayer[1] of the PyTorch framework. This implementation has an output dimensionality d_{val} and an attention dimensionality d_{att} that equals the input dimensionality $d_{\text{x}} = d_{\text{att}} = d_{\text{val}}$. It uses a LayerNorm[4] (LN) transformation that is applied on the *input* of both the Self-Attention (SA) and Feed-Forward (FF) blocks in our configuration. Moreover, it uses residual connections for each block. The output of a single TransformerEncoderLayer TF^l is thus given by

$$\text{TF}^l(\boldsymbol{X}_I) = \boldsymbol{X}_I + \widetilde{\text{FF}}\left(\boldsymbol{X}_I + \widetilde{\text{SA}}\left(\boldsymbol{X}_I\right)\right) \tag{13}$$

with $\widetilde{\text{SA}}(\cdot) = \text{SA}(\text{LN}(\cdot))$ and $\widetilde{\text{FF}}(\cdot) = \text{FF}(\text{LN}(\cdot))$. The SA and FF blocks also contain Dropout layers and the FF uses a GELU activation. The Transformer TF is then simply a composition of N_l Transformer layers: $\text{TF} = \text{TF}^{N_l}\left(\ldots\left(\text{TF}^1(\boldsymbol{X}_I)\right)\right)$.

Typically, the parameters of TF are optimized by computing the loss for a batch of lists. If the lists in a batch do not share the same lengththen all lists are concatenated with feature vectors of zeros $\mathbf{0} = (0)_{i=0}^{d_x}$ until all lists have the same length as the longest list in the batch.

[1] https://pytorch.org/docs/1.13/generated/torch.nn.TransformerEncoderLayer.html

C LABEL SIMULATION DETAILS

In Sec. 6.1, we discuss our simulation of implicit labels on public datasets in three stages: selection, intent and interaction. To ensure reproducibility, we write out the pseudocode of the complete process in Alg. 1.

Note that for the intent T_I, we directly sample from its distribution as given by Eq. 11. However, the actual implicit label y_i is sampled according to a cascade of probability distributions defined in Eq. 12.

Data: Explicit labels $r_i \in \{1, ..., r_{\max}\}$ for item $i \in J$ for a superset of original lists $J \in \mathcal{J}$
Parameters: maximum list length N_s
bootstrapping amount N_b
relevance grade map $\rho : \{1, ..., r_{\max}\} \to [0, 1]$
conversion frequency κ
click noise ϵ
Result: Implicit labels $y_i \in \{0, 1, 2\}$ for item $i \in I$ for a superset of sampled lists $I \in \mathcal{I}$.

$\mathcal{I} \leftarrow \emptyset$
for $J \in \mathcal{J}$ **do**
  /* 1. Selection */
  for $n = 1$ **to** N_b **do**
    if $|J| > N_s$ **then**
      $I \leftarrow \emptyset$
      while $|I| < N_s$ **do**
        $I \leftarrow I \cup \mathcal{U}(J)$
    else
      $I \leftarrow J$

  /* 2. Intent */
  $t \leftarrow \mathcal{U}([0, 1])$
  $r \leftarrow \max_{i \in I} r_i$
  $T_I \leftarrow \begin{cases} 0 & t < 1 - \rho(r) \\ 1 & t \in [1 - \rho(r), (1 - \kappa)\rho(r)] \\ 2 & t > (1 - \kappa)\rho(r) \end{cases}$

  /* 3. Interaction */
  for $i \in I$ **do**
    if $T_I = 2$ **and** $\rho(r_i) > \mathcal{U}([0, 1])$ **then**
      $y_i \leftarrow 2$
    else if $T_I \geq 1$ **and** $\epsilon + (1 - \epsilon)\rho(r_i) > \mathcal{U}([0, 1])$ **then**
      $y_i \leftarrow 1$
    else
      $y_i \leftarrow 0$
  $\mathcal{I} \leftarrow \mathcal{I} \cup I$

Algorithm 1: Implicit label simualtion as described in Sec. 6.1. The $\mathcal{U}(S)$ operator randomly returns a value sampled from set S with a uniform probability distribution.

D HYPERPARAMETERS

Our hyperparameter optimizations (HPOs) were performed separately for the public dataset experiments in Sec. 6 and the Amazon Search experiments in Sec. 7. For the public datasets, we only performed HPOs on Fold 1 of the WEB30K dataset with simulated implicit labels. The hyperparameters with the best NDCG_y on the validation set were then used for the other Folds and for the ISTELLA

Table 5: Hyperparameter selections and ranges for the public dataset experiments. For GBDT, the HPO was later performed separately with 10^3 and 10^4 trees.

		Best		Range
GBDT	`num_iterations`	10^3	10^4	$[10^2, 10^4]$
	`min_data_in_leaf`	255	376	$[10, 500]$
	`num_leaves`	507	271	$[128, 512]$
	`learning_rate`	0.46	0.32	$[0.01, 0.5]$
	`cegb_tradeoff`	0.13	1.65	$[0.1, 10]$
MLP	`lr`	10^{-3}		$\{10^{-4}, 10^{-3}, 10^{-2}\}$
	`weight_decay`	10^{-1}		$\{0, 10^{-2}, 10^{-1}, 1\}$
	`dropout`	0.25		$\{0.1, 0.25, 0.4, 0.5\}$
	`hidden_dims`	$[512, 256, 128]$		$[1, 5]$ layers with size $[128, 1024]$
RankFormer	`nhead`	1		$\{1, 2, 4\}$
	`dim_feedforward`	512		$\{128, 256, 512\}$
	`dropout`	0.25		$\{0.1, 0.25, 0.4, 0.5\}$
	N_l	3		$\{2, 3, 4\}$

and Yahoo datasets. We report all optimized values and their ranges in Tab. 5.

As the hyperparameter space is relatively limited for GBDTs, we conducted a full Bayesian Optimization there, separately for 1,000 and 10,000 trees. We use LightGBM [22] to implement the GBDTs, so we directly list the parameter names as they occur in that framework.

The convergence parameters were first optimized for the MLP, after which they were also used for the RankFormer. In total, we train for 200 epochs and used the Adam optimizer with parameters `lr` and `weight_decay` that starts decaying with an inverse square root schedule after 20 epochs. Additionally, we used `dropout` for the MLP and hidden layer sizes `hidden_dims`.

For the RankFormer, we used the convergence and setting of the MLP. For each layer of its Transformer TF^l (discussed in Appendix B), the `nhead`, `dim_feedforward` and `dropout` were optimized in the PyTorch implementation. The full Transformer component consists of N_l of such layers. To reduce further complexity, we directly choose the h_s and $\boldsymbol{h}_d$ scoring heads to be feed-forward networks with a single 128-dimensional layer.