The LambdaGap Framework for Precision-Oriented Ranking

RAMON ADÀLIA, Lead Molecular Design, S.L., Sant Cugat del Vallès, Spain and Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain
GEMMA SANJUAN and TOMÀS MARGALEF, Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain
ISMAEL ZAMORA, Lead Molecular Design, S.L., Sant Cugat del Vallès, Spain

LambdaRank has proven effective for optimizing information retrieval metrics such as Normalized Discounted Cumulative Gain (NDCG). However, its application to Precision at document k (P@k) poses significant challenges because of the metric's unique definition, which heavily restricts the number of effective training document pairs. This limitation diminishes the learning signal for relevant documents beyond the top k, potentially resulting in suboptimal performance. To overcome this, we propose LambdaGap, a ranking algorithm inspired by LambdaRank specifically tailored for optimizing P@k. LambdaGap replaces the pairwise weighting scheme in LambdaRank by one where pairs of documents within k positions in the ranking are masked out. We establish a theoretical link between LambdaGap and P@k by identifying the implicit metric optimized by the model. Furthermore, we introduce a new metric, Average Relevance Position beyond document k, which can be used in conjunction with LambdaRank to indirectly optimize for P@k. Our extensive experiments on publicly available datasets demonstrate the effectiveness of the proposed methods, yielding statistically significant improvements in P@k performance and highlighting their potential for more efficient training.

CCS Concepts: • **Information systems** → **Learning to rank**; • **Computing methodologies** → *Machine learning*;

Additional Key Words and Phrases: Learning to Rank, LambdaRank, Ranking Metric Optimization

ACM Reference format:
Ramon Adàlia, Gemma Sanjuan, Tomàs Margalef, and Ismael Zamora. 2025. The LambdaGap Framework for Precision-Oriented Ranking. *ACM Trans. Inf. Syst.* 43, 4, Article 97 (June 2025), 39 pages.
https://doi.org/10.1145/3733235

This project is partially supported by the Pla de Doctorats Industrials of the Department of Research and Universities of the Generalitat de Catalunya under contract 2023-DI-00006.
Authors' Contact Information: Ramon Adàlia (corresponding author), Lead Molecular Design, S.L., Sant Cugat del Vallès, Spain and Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain; e-mail: Ramon.Adalia@uab.cat; Gemma Sanjuan, Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain; e-mail: Gemma.Sanjuan@uab.cat; Tomàs Margalef, Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain; e-mail: Tomas.Margalef@uab.cat; Ismael Zamora, Lead Molecular Design, S.L., Sant Cugat del Vallès, Spain; e-mail: ismael.zamora@leadmolecular.com.

ACM 1558-2868/2025/6-ART97
https://doi.org/10.1145/3733235

1 Introduction

Learning-to-Rank (LTR) is a specialized area of machine learning focused on developing algorithms that automatically rank items based on their relevance to a specific query. LTR has broad applications across fields such as information retrieval and natural language processing, including tasks such as web search [30], recommendation systems [9], and keyphrase extraction [14]. The primary objective is to improve the efficiency and accuracy of systems managing large datasets by ensuring that relevant items are prioritized over less relevant ones.

One of the key challenges in LTR is the discrete and non-differentiable nature of many ranking metrics, which makes them unsuitable for direct gradient-based optimization. However, state-of-the-art LTR techniques still employ gradient-based methods either by optimizing surrogate loss functions that approximate the target metric or by building the gradients directly. A notable example of the latter is LambdaRank [2], which was initially developed for artificial neural networks and later adapted for gradient-boosted decision trees through LambdaMART [29].

LambdaRank has been empirically validated for optimizing a range of information retrieval metrics such as **Normalized Discounted Cumulative Gain (NDCG)** [8]. However, its effectiveness for optimizing Precision at k (P@k), a metric that measures the fraction of relevant documents among the top-k ranked items, has not been fully explored.

P@k is widely employed in both classification [15, 22] and LTR [1, 10, 11, 23, 28]. It is the metric of choice in scenarios where the goal is to maximize the number of relevant documents within the top-k positions of a ranking, regardless of their specific order. This situation arises in applications where the top-k documents are selected for subsequent analysis simultaneously rather than sequentially.

A practical example of P@k in an LTR application is found in Multiple Reaction Monitoring [1], an analytical technique used in the pharmaceutical industry to quantify compounds in complex mixtures. This technique requires the pre-selection of fragments of a query compound yielding a high-enough intensity signal in a mass spectrometer. The authors utilize a ranking model to select the top-5 predicted fragments for further analysis, eliminating the need for additional time and compound consumption. Another instance is seen in drug discovery efforts against SARS-CoV-2 [10], where a ranking model narrows down potential drug candidates to the top 100, significantly reducing the number of compounds requiring laboratory testing. In these cases, P@5 and P@100 are the metrics that most faithfully reflect the downstream usage of the retrieval results.

The primary focus of this study is to investigate the optimization of P@k within LTR frameworks. To this end, we evaluate existing LTR algorithms and introduce novel LambdaRank-inspired algorithms specifically designed to enhance P@k performance. We provide a theoretical analysis of these methods, elucidating the specific metrics that each approach optimizes. Furthermore, we validate the effectiveness of the proposed algorithms on three publicly available LTR datasets, demonstrating their superior performance compared with that of LambdaRank.

The remainder of this work is structured as follows: Section 2 provides the necessary background for this study. Section 3 discusses the motivation for our work, discussing the limitations of existing LTR algorithms for optimizing P@k. Section 4 reviews related work and explores potential strategies for improving P@k optimization. Section 5 introduces alternative approaches to LambdaRank for optimizing P@k and presents a theoretical analysis of each method. Section 6 details the experimental setup and reports the results. Finally, Section 7 summarizes the key findings and their implications.

2 Background

In LTR, the training data are constructed from a set of queries Q, each associated with a set of documents. Every document is represented by a feature vector and a relevance label. Given a query

Table 1. Summary of the Notation Used in the Article

Symbol	Description
Q	The set of queries
q	A query
n	The number of documents in a given query
x	The list of feature vectors for a given query
y	The list of labels for a given query
s	The list of scores for a given query
x_i	The feature vector of the document with the ith highest score
y_i	The label of the document with the ith highest score
s_i	The ith highest score
$\mathbb{I}(\cdot)$	The indicator function, returning 1 if the argument is true, 0 otherwise
b_i	Indicator of whether $y_i > 0$, i.e., $\mathbb{I}(y_i > 0)$
$\lvert\cdot\rvert$	The number of elements in a set
σ	A positive hyperparameter used in the pairwise logistic loss function
$l(y, s)$	A loss function evaluated at y and s
M	A ranking metric, e.g., M = NDCG
$\lvert\Delta M\rvert_{ij}$	The weighting factor for the (i, j) pair used in LambdaRank for a metric M
Δ_{ij}	The weighting factor for the (i, j) pair used in LambdaRank-like algorithms
$P(A)$	The probability of event A
$P(A\lvert B)$	The probability of event A given that event B has occurred
$\mathbb{E}[\cdot]$	The expected value of a random variable
$\mathbb{N}$	The natural numbers (not including 0), i.e., $\mathbb{N} = \{1, 2, 3, \ldots\}$
$\mathbb{R}$	The real numbers

$q \in Q$ with n_q documents, we denote the list of feature vectors for all the documents in q by x_q, and the corresponding list of relevance labels by y_q.

The goal of LTR is to find a scoring function Φ that induces an ordering of documents for each query, minimizing a ranking loss. The ordering is obtained by sorting the documents of each query by their decreasing scores. The ranking loss function can then be defined as

$$L(\Phi) = \frac{1}{|Q|} \sum_{q \in Q} l(y_q, \Phi(x_q)),$$

where l is a loss function for a single query, taking the relevance labels and the scores as inputs.

For the remainder of this analysis, we focus on a single query $q \in Q$. To simplify the notation, we drop the subscript q where possible. We also use $s = \Phi(x)$ to denote the list of scores assigned to the documents in the query by Φ. The ith highest score in s is denoted by s_i, with x_i and y_i indicating the corresponding feature vector and relevance label, respectively. We denote the number of documents in q by n. For a summary of the most important mathematical notation used in the remainder of this work, see Table 1.

2.1 LTR Approaches

LTR approaches can be categorized into three groups based on the choice of the loss function: pointwise, pairwise, and listwise losses. For a detailed discussion of these approaches, we refer the reader to [17]. Each category approaches the ranking problem differently:

— *Pointwise approach*: Here, the ranking problem is framed as a regression problem, where any regression loss function can be employed. For example, the mean squared error loss is

defined as:

$$l(y,s) = \frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)^2.$$

This type of loss function evaluates the accuracy of the predicted scores but does not account for the correctness of the relative ordering of the documents.

— *Pairwise approach*: Pairwise loss functions address the LTR task as a pairwise classification problem. The goal is to train a binary classifier that predicts which of two documents is more relevant. RankNet [3] employs the pairwise logistic loss function:

$$l(y,s) = \sum_{y_i > y_j} \log\left(1 + e^{-\sigma(s_i - s_j)}\right),$$

where $\sigma > 0$ is a hyperparameter, typically set to 1.

— *Listwise approach*: Listwise approaches focus on the relative ordering of documents as a whole. A key challenge is that the mapping from scores to ranks, which is commonly used in ranking metrics, is non-differentiable. This makes it difficult to directly optimize these metrics using gradient-based methods. LambdaRank, an extension of RankNet, addresses this by dynamically re-weighting terms in the pairwise loss function. Specifically, it assigns a weight to each document pair based on the absolute change in the ranking metric M that would occur if their scores were swapped, represented as $|\Delta M_{ij}|$. The resulting loss function for LambdaRank is as follows:

$$l(y,s) = \sum_{y_i > y_j} |\Delta M_{ij}| \log\left(1 + e^{-\sigma(s_i - s_j)}\right). \tag{1}$$

We refer to the LambdaRank algorithm applied to a ranking metric M as LambdaRank-M.

LambdaRank has been empirically validated for the direct optimization of various ranking metrics [8]. A commonly used ranking metric is NDCG [13], which is defined as

$$\text{NDCG} = \sum_{i=1}^{n} \frac{2^{y_i} - 1}{\text{IDCG}} \frac{1}{\log_2(i+1)},$$

where IDCG is a normalization factor ensuring that the maximum value of NDCG is 1. This factor is constant for each query. In this context, the change in NDCG when swapping documents i and j is given by

$$|\Delta\text{NDCG}_{ij}| = \frac{2^{y_i} - 2^{y_j}}{\text{IDCG}} \left| \frac{1}{\log_2(i+1)} - \frac{1}{\log_2(j+1)} \right|.$$

Note that the expression for $|\Delta\text{NDCG@}k_{ij}|$ is always positive because in Equation (1) we only sum over pairs where $y_i > y_j$, which implies that $2^{y_i} > 2^{y_j}$, allowing us to omit the absolute value.

The metric of interest in this study, Precision at document k (P@k), measures the fraction of relevant documents within the top-k scoring documents. A relevant document is defined as one with a relevance score greater than 0. Let $b_i = \mathbb{I}(y_i > 0)$, where $\mathbb{I}$ is the indicator function that returns a value of 1 if the condition is true and 0 otherwise. Then

$$|\Delta\text{P@}k_{ij}| = \frac{b_i - b_j}{k} \cdot \mathbb{I}\left(i \le k \oplus j \le k\right),$$

where $\oplus$ denotes the exclusive disjunction operator. In other words, swapping the scores s_i and s_j changes P@k by $1/k$ when exactly one of the two documents is relevant, and exactly one of the two is within the top-k positions. Note that the expression for $|\Delta\text{P@}k_{ij}|$ is always non-negative

because in Equation (1) we only sum over pairs where $y_i > y_j$, which implies that $b_i \geq b_j$, allowing us to omit the absolute value.

For the remainder of this study, we assume that $n > k$, unless stated otherwise. If $n \leq k$, then P@k takes the value of the proportion of relevant documents in the query, and cannot be optimized for.

3 Motivation

LambdaRank has demonstrated effectiveness in optimizing ranking metrics such as NDCG; however, its application to P@k remains underexplored and presents unique challenges. While LambdaRank-P@k inherently aligns with the target metric by treating the document order both within and beyond the top-k positions as irrelevant, this design choice introduces significant limitations. The core issue stems from P@k's definition, where only the presence of relevant documents in the top-k matters, rendering the order within and outside this cutoff irrelevant. Consequently, the weight $|\Delta\text{P@}k_{ij}|$ assigned to document pairs is zero in most cases—specifically, when both documents are either within or outside the top-k. This results in only $k(n-k)$ pairs (where exactly one document is in the top-k) potentially contributing to training signals. When $k \ll n$, this drastically reduces the number of effective training pairs compared with approaches such as RankNet, which utilizes all $n(n-1)/2$ pairs. The binary nature of P@k further compounds this sparsity, as only pairs with one relevant and one irrelevant document contribute, excluding pairs with matching relevance labels.

The scarcity of meaningful training signals has concrete implications. During training, relevant documents initially ranked outside the top-k receive gradients only from the limited number of irrelevant documents within the top-k, limiting their ability to ascend in the ranking. In contrast, methods such as RankNet leverage gradients from all document pairs, enabling more comprehensive learning. This discrepancy can lead to slower convergence and suboptimal performance for LambdaRank-P@k [8], particularly in early training stages when relevant documents may frequently reside outside the top-k. Moreover, while increasing the training data size might mitigate some issues, the intrinsic challenge of sparse gradients remains.

Despite these limitations, LambdaRank-P@k retains a critical advantage: As demonstrated in Section 5.2, it directly optimizes a bound for P@k. Alternative approaches, such as LambdaRank-NDCG, optimize surrogate objectives (e.g., NDCG) that may diverge from P@k's goals. For example, LambdaRank-NDCG might prioritize the ranking of a single highly relevant document over multiple moderately relevant ones, potentially lowering P@k. Thus, while LambdaRank-P@k faces theoretical constraints, it remains a viable candidate for tasks where alignment with P@k is paramount.

To address these challenges, an ideal method should satisfy two key criteria:

(1) *Theoretical grounding*: Direct optimization of P@k or a provably related surrogate.
(2) *Extended pairwise utilization*: Exploitation of pairwise signals across the entire ranked list, including those beyond the top-k, without violating P@k's indifference to the intra-top-k order.

Existing methods fall short of meeting both requirements:

— *LambdaRank-P@k* meets the first criterion but ignores pairwise interactions beyond the top-k, limiting gradient diversity.
— *RankNet and LambdaRank-NDCG* utilize full-list pairs but optimize unproven surrogates for P@k.
— *Hybrid approaches* that combine P@k-specific gradients with auxiliary losses degrade theoretical guarantees through unvalidated approximations.

This methodological gap motivates our work: We seek a solution that preserves LambdaRank-P@k's theoretical alignment with P@k while harnessing pairwise signals from the entire list. By bridging this divide, our approach aims to increase training efficiency and model performance without compromising metric-specific optimization. The subsequent sections analyze this objective's feasibility and propose novel methods to achieve it.

4 Related Work

4.1 Metric-Specific Optimization

Research on the direct optimization of ranking metrics primarily involves two approaches: approximation methods and dynamic re-weighting of document pairs. Approximation methods involve constructing a differentiable loss function by approximating the ranking operator π, which maps document scores to positions [6, 25]. For instance, ApproxNDCG [25] employs the approximation

$$\tilde{\pi}(s_i|s) = 1 + \sum_{j=1}^{n} \text{sigmoid}(s_i - s_j),$$

where $\text{sigmoid}(x) = (1 + \exp(-\sigma x))^{-1}$ and $\sigma > 0$ is a hyperparameter. Setting σ to a sufficiently large value leads to a close approximation of the original ranking operator. However, the more accurate the approximation is, the more "non-smooth" the loss function becomes, which may render it difficult to optimize, potentially requiring the use of more complex techniques such as simulated annealing or random optimization [17].

Instead, we focus on the dynamic re-weighting approach, with LambdaRank being a notable example. Although LambdaRank has demonstrated empirical success in optimizing various information retrieval metrics, the theoretical basis for its effectiveness was not fully addressed in the original paper. The *LambdaLoss* framework [27] provides a theoretical foundation for LambdaRank, offering a general methodology for directly optimizing ranking metrics defined by gain and discount functions, such as NDCG. However, the original work omits critical assumptions necessary for extending its applicability to other metrics, such as P@k.

A detailed discussion of LambdaLoss is provided in Section 5, where we bridge this gap by presenting a more generalized formulation that includes a comprehensive list of required assumptions. This extension facilitates its application to a broader range of LTR metrics. Furthermore, we demonstrate how these insights integrate with the proposed LambdaGap framework, enabling improved metric-specific optimization.

4.2 Optimization of Truncated Metrics

Although LambdaRank is typically applied to NDCG, in practice, it is often used for truncated variants such as NDCG@k, which is motivated by an emphasis on top-k documents and computational efficiency, as truncation significantly reduces the number of considered document pairs. However, LambdaRank-NDCG@k suffers from the same limitation as LambdaRank-P@k: Document pairs outside the cutoff do not contribute to gradient calculations. Addressing this limitation could yield valuable improvement strategies for optimizing P@k.

Recent work has focused on enhancing training signals for truncated metrics. In [19], the authors introduce the concept of *gradient incoherence* in LambdaRank when applied to NDCG@k, illustrating scenarios where LambdaRank assigns a higher gradient to a less relevant document ranked above a more relevant one. This misalignment can exacerbate incorrect pairwise orderings. The study identifies the interplay between metric discounts and truncation as a primary cause of gradient incoherence. Specifically, when both documents in a pair fall outside the top-k positions, they are excluded from gradient calculations, leading to potentially misleading training signals.

To address this issue, the authors propose *Lambda-eX*, a method that incorporates certain excluded document pairs into the gradient calculation through various strategies. This approach enhances training signals for NDCG while retaining the computational advantages of LambdaRank when applied to NDCG@k. Lambda-eX expands the set of gradient-relevant document pairs in LambdaRank-NDCG@k by including a *full-gradient document set*, which forces some documents outside the top-k rankings to be compared with all other documents for the query. The authors define two key types of documents:

— *False top-k documents*: Documents incorrectly ranked within the top-k positions. In a perfect ranking, no document should fall into this category.
— *Missed top-k documents*: Relevant documents incorrectly ranked below the top k. In a perfect ranking, a document can fall into this category if relevance ties are present.

The study proposes three strategies for constructing the full-gradient document set:

— `static`: For each false top-k document, include a missed top-k document, starting with the better-ranked ones.
— `random`: For each false top-k document, randomly sample a missed top-k document.
— `all`: Include all missed top-k documents.

Two hybrid strategies, `all-static` and `all-random`, extend `all` by reverting to other strategies when all relevant documents would be included in the full-gradient document set, which occurs when the smallest positive relevance label in the query coincides with the smallest relevance label in the missed top-k documents.

Experimental results demonstrate that Lambda-eX reduces gradient incoherence and significantly improves the NDCG. However, this approach is incompatible with P@k, as Lambda-eX bases its gradient calculations on the non-truncated metric NDCG, so that a pair of documents below the cutoff value can contribute to the gradient. Unlike NDCG@k, however, P@k is not a truncated version of a ranking metric. Its non-truncated counterpart, the count of relevant documents, is independent of rankings. Consequently, including document pairs outside the top-k in gradient calculations does not impact P@k, as swapping such pairs does not alter the metric.

Similarly, the method in [12] is not directly applicable to P@k. The authors adapt the LambdaLoss framework for NDCG@k by introducing a correction multiplier for document pairs involving at least one document outside the top k:

$$\mu_{ij} = \left(1 - \frac{1}{D(r)}\right)^{-1},$$

where $D(r)$ is the discount of the lower-ranked document in the pair ($r = \max\{i, j\}$). While this approach improves NDCG@k, extending it to P@k is non-trivial. Again, the method requires the existence of LambdaLoss applied to the non-truncated version of P@k, which does not exist. Moreover, while P@k can be expressed using gain and discount functions with infinite discounts for positions beyond k, this trivializes the correction multiplier, effectively reducing the method to standard LambdaLoss.

5 Optimization Alternatives for P@k

In this section, we introduce alternative approaches for optimizing P@k. All the proposed methods fall within the class of *LambdaRank-like algorithms*, which we define as the algorithms that are

characterized by a loss function of the form

$$l(y, s) = \sum_{y_i > y_j} \Delta_{ij} \log\left(1 + e^{-\sigma(s_i - s_j)}\right),$$

where the weighting factor Δ_{ij} is a key component that may depend on the relevance labels y_i and y_j and the scores s_i and s_j. Each proposed approach modifies this weighting factor to potentially better align with the optimization of P@k.

5.1 LambdaGap

One defining characteristic of P@k is that the order of the top-k ranked documents does not matter. This property is reflected in the weighting factor used in LambdaRank-P@k, as $|\Delta\text{P@}k_{ij}| = 0$ whenever both $i \leq k$ and $j \leq k$. We propose extending this concept to a weighting factor that disregards the order within any set of k consecutively ranked documents. In this way, the intra-top-k ordering of documents continues to be irrelevant, but document pairs outside the cutoff may contribute to the gradient.

To achieve this, consider a function $f : \mathbb{N} \to [0, +\infty)$ such that $f(i) = 0$ for all $i < k$. We define the LambdaRank-like algorithm induced by this function f as *LambdaGap*, with a weighting factor of the form

$$\Delta_{ij} = \frac{b_i - b_j}{k} \cdot f(|i - j|).$$

A simple choice for f is $f : |i - j| \mapsto \mathbb{I}(|i - j| \geq k)$, where all pairs of documents that are not within a window of k consecutively ranked documents are weighted equally. We refer to the LambdaRank-like algorithm using this weighting factor as *LambdaGap-X*. Another possible choice for f is $f : |i - j| \mapsto \mathbb{I}(|i - j| = k)$, where only pairs that are exactly k positions apart in the ranking receive non-zero weight. This approach is referred to as *LambdaGap-S*.

While these constructions of weighting factors may seem arbitrary and particularly restrictive, the remainder of this section establishes a theoretical connection between LambdaGap and P@k; specifically, we demonstrate that LambdaGap maximizes a lower bound for a metric closely related to P@k.

5.1.1 Connection with P@k. To establish a connection between LambdaGap and P@k, we draw upon the LambdaLoss framework [27]. In that work, it was shown that LambdaRank-like algorithms can be interpreted as instances of the **Expectation-Maximization (EM)** algorithm.

An EM algorithm is an iterative method for finding local maximum likelihood estimates of statistical models, especially when hidden (latent) variables are present. It equivalently finds a local minimum of the negative log-likelihood function. The EM algorithm alternates between an expectation step (E-step), where a likelihood function is constructed based on the current parameter estimates, and a maximization step (M-step), where the parameters that maximize this likelihood function are computed. While the EM algorithm is guaranteed to converge to a local maximum of the likelihood function, it may not always find the global maximum. The quality of the solution often depends on the initial parameter estimates.

To formulate LambdaRank-like models within the EM framework, we need to construct a likelihood function. This is achieved by defining the probability of observing the data y given the scores s as

$$P(y|s) = \sum_{\pi \in \Pi} P(y|s, \pi) P(\pi|s),$$

where Π represents the space of possible permutations of y. The distribution of these permutations conditioned on the scores s acts as the latent variable in the EM algorithm. In the E-step, we estimate

this distribution based on the current scores s. LambdaRank-like models utilize a hard assignment distribution, defined as

$$P(\pi|s) = \begin{cases} 1 & \text{if } \pi = \hat{\pi}, \\ 0 & \text{otherwise,} \end{cases}$$

where $\hat{\pi}$ is the permutation of y that sorts the documents in decreasing order of their scores. Additionally, we define

$$P(y_i > y_j|s, \pi) = \left(\frac{1}{1 + e^{-\sigma(s_i - s_j)}}\right)^{\Delta_{ij}},$$

where Δ_{ij} depends on the specific permutation π, even though it is not explicitly indicated, but remains independent of s. This formulation leads to the following negative log-likelihood function:

$$\begin{aligned} l(y, s) &= -\sum_{y_i > y_j} \log \sum_{\pi \in \Pi} P(y_i > y_j|s, \pi) P(\pi|s) \\ &= -\sum_{y_i > y_j} \log P(y_i > y_j|s, \hat{\pi}) \\ &= \sum_{y_i > y_j} \Delta_{ij} \log\left(1 + e^{-\sigma(s_i - s_j)}\right). \end{aligned}$$

The negative log-likelihood described here coincides with the loss function used in LambdaRank-like models. During the M-step, the scores s are re-estimated by minimizing this function, typically through a gradient descent step. This method is not a strict implementation of EM but still serves as a coordinate descent method in the negative log-likelihood function, provided that the step size is sufficiently small.

The only remaining component of the algorithm is the initialization of s. In the original RankNet and other neural-network-based architectures, this can be achieved through random initialization. For tree-based methods, such as LambdaMART [4], initialization is performed by fitting a decision tree regressor to the relevance labels via a least-squares objective.

We now present the following lemma:

Lemma 5.1. *Let $M = \sum_{i=1}^{n} f(y_i)g(i)$, where f and g are real-valued functions over $\mathbb{N}$, f is non-decreasing and g is non-increasing. Let $\delta : \mathbb{N}^2 \to [0, +\infty)$ be defined by*

$$\delta(i, j) = g(|i - j|) - g(|i - j| + 1).$$

Then, the LambdaRank-like model with weights $\Delta_{ij} = \big(f(y_i) - f(y_j)\big)\,\delta(i, j)$ maximizes a lower bound for M.

The following proof is an adaptation of the derivation for NDCG-Loss2 in [27].

Proof. First, note that maximizing a lower bound for M is equivalent to minimizing an upper bound for $-M$. Let $C_1 = g(1)\sum_{i=1}^{n} f(y_i)$. Since C_1 is a constant, maximizing a lower bound for $-M$ is equivalent to maximizing a lower bound for $C_1 - M$. Then,

$$C_1 - M = g(1)\sum_{i=1}^{n} f(y_i) - \sum_{i=1}^{n} f(y_i) \cdot g(i) = \sum_{i=1}^{n} f(y_i) \cdot (g(1) - g(i)).$$

Now we use that, $g(1) - g(i) = \sum_{j=1}^{i-1} \delta(i, j)$, which is easily verified by expanding the sum and the definition of δ. Then, the expression for $C_1 - M$ becomes

$$\begin{aligned} C_1 - M &= \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} \delta(i, j) \\ &= \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{n} \delta(i, j) \mathbb{I}(s_i < s_j) \\ &= \frac{1}{2} \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{n} \delta(i, j) \mathbb{I}(s_i < s_j) + \frac{1}{2} \sum_{j=1}^{n} f(y_j) \sum_{i=1}^{n} \delta(j, i) \mathbb{I}(s_j < s_i) \\ &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \delta(i, j) \left[f(y_i) \mathbb{I}(s_i < s_j) + f(y_j) \mathbb{I}(s_j < s_i) \right], \end{aligned}$$

where the second equality holds because $s_i < s_j$ if and only if $j < i$, since s is sorted in decreasing order, and the last equality holds because $\delta(i, j) = \delta(j, i)$ for all $i, j \in \mathbb{N}$. We now split the sum into three sums, depending on if $y_i > y_j$, if $y_i < y_j$ or if $y_i = y_j$. Due to the symmetry of the summand with respect to swapping i and j, the first two sums are equivalent. The third sum, where $y_i = y_j$, simplifies to

$$C_2 = \frac{1}{2} \sum_{y_i = y_j} \delta(i, j) f(y_j) [\mathbb{I}(s_i < s_j) + \mathbb{I}(s_j < s_i)] = \frac{1}{2} \sum_{y_i = y_j} \delta(i, j) f(y_j),$$

since $f(y_i) = f(y_j)$. Hence,

$$\begin{aligned} C_1 - M &= \sum_{y_i > y_j} \delta(i, j) \left[f(y_i) \mathbb{I}(s_i < s_j) + f(y_j) \mathbb{I}(s_j < s_i) \right] + C_2 \\ &= \sum_{y_i > y_j} \delta(i, j) \left[f(y_i) \mathbb{I}(s_i < s_j) + f(y_j)(1 - \mathbb{I}(s_i < s_j)) \right] + C_2 \\ &= \sum_{y_i > y_j} \delta(i, j) \left(f(y_i) - f(y_j) \right) \mathbb{I}(s_i < s_j) + C_2 + C_3, \\ &\leq \sum_{y_i > y_j} \Delta_{ij} \log_2 \left(1 + e^{-\sigma(s_i - s_j)} \right) + C_2 + C_3, \end{aligned}$$

where $C_3 = \sum_{y_i > y_j} \delta(i, j) f(y_j)$. In the last step, we used that

$$\mathbb{I}(x < 0) \leq \log_2 (1 + e^{-\sigma x}) \iff \mathbb{I}(s_i < s_j) \leq \log_2 \left(1 + e^{-\sigma(s_i - s_j)} \right).$$

To see this, note that $h(x) = \log_2 (1 + e^{-\sigma x})$ is positive, strictly decreasing, and $h(0) = 1$, which implies that $h(x) \geq 1$ for all $x \leq 0$.

Note that C_1, C_2, and C_3 are all constant in the M-step of the EM algorithm, as they do not depend on s, given a fixed permutation π of y determined in the E-step, and log and $\log_2$ differ only by a multiplicative constant. It follows from the theoretical guarantees of EM algorithms that the LambdaRank-like algorithm with the weighting factor Δ_{ij} maximizes a lower bound for M. □

We can now leverage this result to establish a connection between LambdaGap and P@k.

Proposition 5.2 (LambdaGap metric). *Let $f : \mathbb{N} \to [0, +\infty)$ such that $f(i) = 0$ for all $i < k$. Then, the LambdaGap algorithm induced by f maximizes a lower bound for*

$$M = \mathrm{P@}k + \frac{1}{k} \sum_{i=k+1}^{n} b_i \left[1 - \sum_{j=k}^{i-1} f(j) \right].$$

Proof. Let $C = \sum_{i=1}^{n} b_i/k$. Since C is a constant, maximizing a lower bound for M is equivalent to maximizing a lower bound for $M - C$. Then,

$$\begin{aligned}
M - C &= \mathrm{P@}k + \frac{1}{k} \sum_{i=k+1}^{n} b_i \left[1 - \sum_{j=k}^{i-1} f(j) \right] - C \\
&= \frac{1}{k} \sum_{i=1}^{k} b_i + \frac{1}{k} \sum_{i=k+1}^{n} b_i - \frac{1}{k} \sum_{i=k+1}^{n} b_i \sum_{j=k}^{i-1} f(j) - C \\
&= C - \frac{1}{k} \sum_{i=k+1}^{n} b_i \sum_{j=k}^{i-1} f(j) - C \\
&= -\frac{1}{k} \sum_{i=k+1}^{n} b_i \sum_{j=k}^{i-1} f(j) \\
&= -\frac{1}{k} \sum_{i=1}^{n} b_i \sum_{j=1}^{i-1} f(j),
\end{aligned}$$

where the last equality holds because $f(i) = 0$ for all $i < k$, which implies that $\sum_{j=1}^{i-1} f(j) = 0$ for all $i \leq k$. Let $h(y_i) = b_i/k$ and $g(i) = -\sum_{j=1}^{i-1} f(j)$. Note that g is non-increasing because f only outputs non-negative numbers real numbers. Then, $M - C = \sum_{i=1}^{n} h(y_i)g(i)$, and

$$\delta(i, j) = g(|i - j|) - g(|i - j| + 1) = f(|i - j|).$$

It follows from Lemma 5.1 that the LambdaRank-like algorithm with the weighting factor

$$\Delta_{ij} = \big(h(y_i) - h(y_j)\big)\, \delta(i, j) = \frac{b_i - b_j}{k} \cdot f(|i - j|)$$

maximizes a lower bound for $M - C$ and, hence, for M. □

Proposition 5.2 highlights the relationship between LambdaGap and P@k, demonstrating that LambdaGap optimizes a bound on P@k with an added penalty term for documents ranked outside the top k. The magnitude of this penalty is determined by the choice of f. Importantly, within the top k, the order of the documents remains irrelevant to the derived metric M. We now present the results for LambdaGap-X and LambdaGap-S.

Corollary 5.3 (LambdaGap-S metric). *LambdaGap-S maximizes a lower bound for P@k.*

Proof. The weighting function in LambdaGap-S is induced by the function $f : |i-j| \mapsto \mathbb{I}(|i-j| = k)$. It is enough to notice that, for all $i > k$,

$$1 - \sum_{j=k}^{i-1} f(j) = 1 - \sum_{j=k}^{i-1} \mathbb{I}(j = k) = 1 - \mathbb{I}(k = k) = 0,$$

from which

$$M = \text{P@}k + \frac{1}{k}\sum_{i=k+1}^{n} b_i \left[1 - \sum_{j=k}^{i-1} f(j)\right] = \text{P@}k.$$

□

Corollary 5.3 establishes LambdaGap-S as the most natural variant of LambdaGap for P@k optimization. Moreover, it satisfies both conditions presented in Section 3, i.e., provably directly optimizing P@k and utilizing pairwise interactions between documents beyond the cutoff of k documents, making it particularly appealing from a theoretical perspective.

However, some practical limitations emerge:

(1) *Gradient sparsity*: The restrictive weighting scheme in LambdaGap-S likely creates fewer effective training pairs than even LambdaRank-P@k does, revealing that our Section 3 criteria may not be sufficient for optimal learning dynamics.
(2) *Pathological convergence*: Consider a worst-case ranking where relevant documents are periodically spaced ($b_i = \mathbb{I}(i \equiv -1 \mod k)$). Here, P@$k$ remains suboptimal ($1/k$) yet the gradients vanish entirely ($l \equiv 0$ during the M-step), halting training progress. No such thing would occur with LambdaRank-P@k. Nevertheless, real-world datasets rarely exhibit such pathological structures, and our experiments in Section 6 demonstrate that LambdaGap-S remains viable despite these theoretical caveats.

Corollary 5.4 (LambdaGap-X metric). *LambdaGap-X minimizes an upper bound for the* ***Average Relevance Position (ARP)*** *beyond document k ($\text{ARP}_b k$), defined as*

$$\text{ARP}_b k = \sum_{i=1}^{n-k} b_{i+k} \cdot i = \sum_{i=k+1}^{n} b_i \cdot (i-k).$$

This metric is closely related to the ARP [16], defined as $\text{ARP} = \sum_{i=1}^{n} b_i \cdot i$. The ARP can be viewed as a special case of the $\text{ARP}_b k$ when $k = 0$. Intuitively, the $\text{ARP}_b k$ metric penalizes the presence of relevant documents outside the top k, with the penalty increasing linearly with their distance from it.

Despite some similarity to truncated metrics, $\text{ARP}_b k$ differs in that it is *reverse-truncated*, meaning that it only considers documents ranked below the top k, as seen in its definition. Hence, while the $\text{ARP}_b k$ maintains the property that the order of documents within the top k does not affect the metric, the order beyond the top k does matter. This distinction may incentivize pushing relevant documents closer to the top k. Additionally, LambdaGap-X avoids the theoretical issues associated with LambdaGap-S, even though it may diverge from the true target metric.

Proof. Using the same notation from the proof of Proposition 5.2, we have already shown that

$$M - C = -\frac{1}{k}\sum_{i=k+1}^{n} b_i \sum_{j=k}^{i-1} f(j).$$

If $f(i) = \mathbb{I}(i \geq k)$ as in LambdaGap-X, then $\sum_{j=k}^{i-1} f(j) = i - k$ when $i > k$. It follows that LambdaGap-X maximizes a lower bound for

$$M - C = -\frac{1}{k}\sum_{i=k+1}^{n} b_i (i-k) = -\frac{1}{k}\text{ARP}_b k,$$

which is equivalent to minimizing an upper bound for the $\text{ARP}_b k$. □

We conclude this section by presenting an additional result, which follows directly from Corollary 5.4. This comes from the realization that, when $k = 1$, LambdaGap-X is very similar to RankNet. In fact, it coincides with RankNet if the relevance labels are binary. We refer to the special case of LambdaGap-X when $k = 1$ as *BinRankNet*, given that it is a binary version of RankNet. The following result offers insight into the metric optimized by BinRankNet.

Corollary 5.5 (BinRankNet metric). *BinRankNet minimizes an upper bound for the ARP.*

Proof. Since BinRankNet can be seen as LambdaGap-X with $k = 1$, BinRankNet minimizes an upper bound for $M = \text{ARP}_b 1$. Let $C = \sum_{i=1}^{n} b_i$. Then,

$$C + M = \sum_{i=1}^{n} b_i + \sum_{i=2}^{n} b_i \cdot (i-1) = b_1 + \sum_{i=2}^{n} b_i \cdot i = \sum_{i=1}^{n} b_i \cdot i = \text{ARP}.$$

□

5.2 LambdaRank-ARP$_b k$

While investigating the metric optimized by LambdaGap, we have stumbled upon the $\text{ARP}_b k$ metric, which ignores the specific ordering within the top-k documents but still incentivizes pushing relevant documents towards the top k. Instead of using LambdaGap-X, one may consider employing LambdaRank directly on this metric. In this case, the weighting factor would be

$$|\Delta \text{ARP}_b k_{ij}| = (b_i - b_j)\,|(i-k)\cdot \mathbb{I}(i \geq k) - (j-k)\cdot \mathbb{I}(j \geq k)|\,.$$

This may benefit the training process, as more pairs of documents will have a chance to contribute to the training while not deviating excessively from the target metric. Naturally, the question arises as to which metric is actually optimized by this approach. Once again, we lean on the work of [27] and provide a slightly more general version of their bound for the metric optimized by LambdaRank.

Lemma 5.6 [LambdaRank metric]. *Let $M = \sum_{i=1}^{n} f(y_i)g(i)$, where f and g are non-decreasing real-valued functions over $\mathbb{N}$. Then, LambdaRank-M minimizes an upper bound for the metric*

$$M' = \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} (g(i) - g(j))\,.$$

Proof. Let $\delta(i,j) = |g(i) - g(j)|$. Then, $M' = \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} \delta(i,j)$. Following the same derivation from Lemma 5.1, we find that

$$M' \leq \sum_{y_i > y_j} \delta(i,j)\,\big(f(y_i) - f(y_j)\big) \log_2\left(1 + e^{-\sigma(s_i - s_j)}\right) + C_2 + C_3,$$

where $C_2 = \sum_{y_i = y_j} \delta(i,j) f(y_j)/2$, and $C_3 = \sum_{y_i > y_j} \delta(i,j) f(y_j)$. Since C_2 and C_3 are constant in the M-step of the EM algorithm, it follows that the LambdaRank-like algorithm with weighting factor

$$\Delta_{ij} = \delta(i,j)\,\big(f(y_i) - f(y_j)\big) = |\Delta M_{ij}|$$

minimizes an upper bound for M'. □

Corollary 5.7 (LambdaRank-ARP$_b k$ metric). *LambdaRank-ARP$_b k$ minimizes an upper bound for the metric*

$$M = \text{ARP}_b k + \frac{1}{2k} \sum_{i=1}^{n-k} b_{i+k} \cdot i(i-1).$$

Proof. Let $f(y_i) = b_i$ and $g(i) = (i-k)\mathbb{I}(i > k)$, so that $\text{ARP}_b k = \sum_{i=1}^{n} f(y_i)g(i)$. Let $i = k + m$, where m is a positive natural number. Then, $g(i) = m$, $g(j) = 0$ if $j \leq k$ and $g(j) = (j-k)$ if $j > k$, and

$$\begin{aligned}
\sum_{j=1}^{i-1} (g(i) - g(j)) &= \sum_{j=1}^{k+m-1} (m - g(j)) \\
&= \sum_{j=1}^{k} (m - g(j)) + \sum_{j=k+1}^{k+m-1} (m - g(j)) \\
&= km + \sum_{j=k+1}^{k+m-1} (m - j + k) \\
&= km + \sum_{j=1}^{m-1} (m - j) \\
&= km + \sum_{j=1}^{m-1} j \\
&= km + \frac{m(m-1)}{2} \\
&= k(i-k) + \frac{(i-k)(i-k-1)}{2}.
\end{aligned}$$

On the other hand, if $i \leq k$, then $\sum_{j=1}^{i-1} (g(i) - g(j)) = 0$, as all terms involved in the sum are 0. Let $M' = \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} (g(i) - g(j))$. This construction satisfies the requirements of Lemma 5.6. Moreover,

$$\begin{aligned}
\frac{1}{k} M' &= \frac{1}{k} \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} (g(i) - g(j)) \\
&= \frac{1}{k} \sum_{i=k+1}^{n} f(y_i) \sum_{j=1}^{i-1} (g(i) - g(j)) \\
&= \sum_{i=k+1}^{n} b_i \cdot (i-k) + \frac{1}{2k} \sum_{i=k+1}^{n} b_i \cdot (i-k)(i-k-1) \\
&= \text{ARP}_b k + \frac{1}{2k} \sum_{i=1}^{n-k} b_{i+k} \cdot i(i-1) = M.
\end{aligned}$$

Hence, LambdaRank-ARP$_b k$ minimizes an upper bound M. □

Corollary 5.7 shows that LambdaRank-ARP$_b k$ optimizes for a bound of ARP$_b k$, with an additional term that includes a quadratic penalization term for relevant documents outside the top k. Whether this is desirable for P@k will be assessed experimentally.

We conclude this section with a similar analysis for LambdaRank-P@k.

Corollary 5.8 (LambdaRank-P@k metric). *LambdaRank-P@k maximizes a lower bound for P@k.*

Proof. $f(y_i) = \frac{1}{k} b_i$, $g(i) = \mathbb{I}(i > k)$. Let $i = k + m$, where m is a positive natural number. Then, $g(i) = 1$, $g(j) = 0$ if $i \leq k$ and $g(j) = 1$ if $i > k$, and

$$\begin{aligned}\sum_{j=1}^{i-1} (g(i) - g(j)) &= \sum_{j=1}^{k+m-1} (1 - g(j)) \\ &= \sum_{j=1}^{k} (1 - g(j)) + \sum_{j=k+1}^{k+m-1} (1 - g(j)) \\ &= \sum_{j=1}^{k} 1 + \sum_{j=k+1}^{k+m-1} (1 - 1) \\ &= k.\end{aligned}$$

On the other hand, if $i \leq k$, then $\sum_{j=1}^{i-1} (g(i) - g(j)) = 0$, as all terms involved in the sum are 0. Let $M' = \sum_{i=1}^{n} f(y_i) \sum_{j=1}^{i-1} (g(i) - g(j)) = k \sum_{i=1}^{n} f(y_i) \mathbb{I}(i > k)$. This construction satisfies the requirements of Lemma 5.6. Hence, LambdaRank-P@k maximizes an upper bound for M'.

Let $C = \sum_{i=1}^{n} f(y_i)$. Since C is a constant, maximizing an upper bound for M' is equivalent to minimizing an upper bound for $M = C - M'/k$. Moreover,

$$M = C - \frac{M'}{k} = \sum_{i=1}^{n} f(y_i)[1 - \mathbb{I}(i > k)] = \sum_{i=1}^{n} f(y_i) \mathbb{I}(i \leq k) = \text{P@}k.$$

□

5.3 Binarizing

When using LambdaRank-NDCG, the model may sometimes prioritize placing fewer highly relevant documents in the top-k positions over more moderately relevant documents. This is because such an arrangement can yield a higher NDCG score, even though it may result in a lower P@k. To address this issue, one potential strategy is to convert the relevance labels to a binary format, setting all positive labels to 1 and the rest to 0. An equivalent approach is to modify LambdaRank's target metric to use a binarized version of the NDCG. For instance, the **Binarized NDCG (BNDCG)** can be defined as

$$\text{BNDCG} = \sum_{i=1}^{n} \frac{2^{b_i} - 1}{\text{IBDCG}} \frac{1}{\log_2(i + 1)},$$

where IBDCG is a normalization factor ensuring that the maximum value of BNDCG is 1.

While binarizing objectives can be helpful when optimizing for binary metrics, it does not guarantee improved performance. Binarization can lead to the loss of valuable information that might be beneficial for the learning process, even if the optimization target is not explicitly based on non-binary labels. Therefore, the impact of this approach requires empirical evaluation to determine its effectiveness.

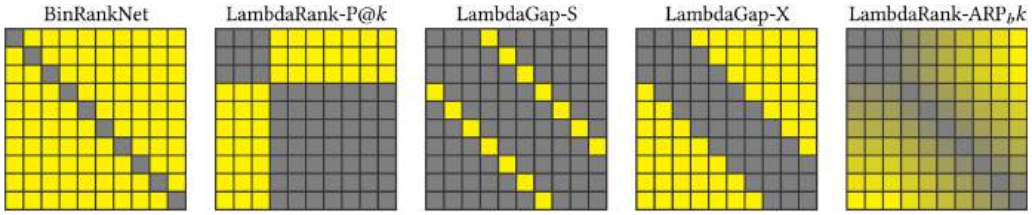

Fig. 1. Impact of $\delta(i, j)$ on pair selection across different algorithms for a query with $n = 10$ documents, in the form of heatmaps. For each algorithm, the opacity of yellow in the cell at the ith row and jth column represents the magnitude of $\delta(i, j)$ for the pair formed by documents at positions i and j in the current ranking. Gray cells indicate pairs that are excluded from the training process.

5.4 Combining Methods

All the methods discussed thus far belong to the class of LambdaRank-like models, where the weighting factor takes the form $\Delta_{ij} = (b_i - b_j)\delta(i, j)$, with $\delta : \mathbb{N}^2 \to \mathbb{R}$, up to a multiplicative constant. The term $b_i - b_j$ is equivalent to $\mathbb{I}(b_i \neq b_j)$ and functions as an indicator of pairs where exactly one document is relevant. When binary, the term $\delta(i, j)$ acts as a masking factor, effectively deciding which pairs should be excluded from the training process at a given step. This binary masking behavior is characteristic of all the methods discussed, with the exception of LambdaRank-ARP$_b$k, where $\delta(i, j)$ is not merely a mask but a weighting function that assigns different importance levels to each document pair. The varying impacts of the choices of δ functions on pair selection are visually represented in Figure 1.

Given the structural similarities among the proposed methods, one might consider hybrid approaches that combine multiple techniques. A similar concept appears in [27], where LambdaRank-NDCG and LambdaLoss-NDCG are merged to form a hybrid model. Specifically, given two LambdaRank-like approaches A and B with weighting factors Δ^A_{ij} and Δ^B_{ij}, a hybrid approach can be defined as

$$\Delta_{ij} = \Delta^A_{ij} + \mu \cdot \Delta^B_{ij}, \tag{2}$$

where μ is a fixed positive real number. If A and B optimize upper bounds for metrics M_A and M_B, their hybrid method optimizes an upper bound of $M_A + \mu M_B$. The same reasoning applies to algorithms optimizing lower metric bounds, with multiplication by -1.

In particular, combining LambdaGap-S with LambdaRank-P@k results in a hybrid approach that optimizes P@k while incorporating pairwise interactions beyond the document cutoff k. This approach satisfies the criteria in Section 3 and resolves the theoretical issues associated with LambdaGap-S, as discussed in Corollary 5.3. Notably, this hybridization enhances pairwise interactions without sacrificing direct metric optimization, an advantage that is less apparent for NDCG-based objectives.

We consider the following combinations:

(1) LambdaGap-S+: Hybrid of LambdaRank-P@k and LambdaGap-S.
(2) LambdaGap-X+: Hybrid of LambdaRank-P@k and LambdaGap-X.
(3) LambdaGap-S++: Hybrid of LambdaRank-ARP$_b$k and LambdaGap-S.
(4) LambdaGap-X++: Hybrid of LambdaRank-ARP$_b$k and LambdaGap-X.

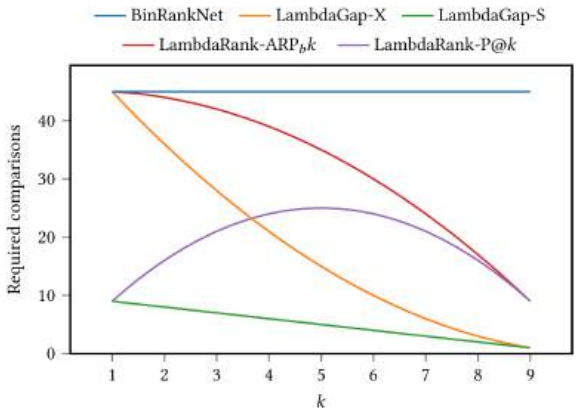

Fig. 2. Number of required pairwise comparisons for different algorithms as the value of k varies, with $n = 10$.

5.5 Complexity Analysis

Different algorithms offer varying opportunities for optimization based on their definitions. For LambdaRank-like models that use gradient descent, optimizations can be made during the gradient calculation steps, particularly in how Δ_{ij} is constructed. Typically, computing gradients for LambdaRank-like models involves ordering the data according to the scores and performing additional computations.

When dealing with binary weighting factors of the form $\Delta_{ij} = (b_i - b_j)\delta(i, j)$, where $\delta(i, j)$ is a binary function that outputs either 0 or 1, significant computational savings can be achieved by skipping pairs where $\delta(i, j) = 0$. This selective computation reduces the number of pairs to evaluate, which is particularly beneficial in scenarios where the number of possible document pairs is large.

The number of required pairwise comparisons varies significantly across different algorithms, and can be visualized in Figure 2. The number of required pairwise comparisons is as follows:

(1) BinRankNet: $n(n-1)/2$,
(2) LambdaRank-ARP$_b k$: $n(n-1)/2 - k(k-1)/2$,
(3) LambdaGap-X: $(n-k)(n-k+1)/2$,
(4) LambdaRank-P@k: $k(n-k)$,
(5) LambdaGap-S: $n-k$.

BinRankNet always requires the most pairwise comparisons, matching those of regular RankNet. The complexity decreases in the order

$$\begin{aligned} &\text{BinRankNet} \succeq \text{LambdaRank-ARP}_b k \succeq \text{LambdaRank-P@}k \succeq \text{LambdaGap-S}, \\ &\text{BinRankNet} \succeq \text{LambdaRank-ARP}_b k \succeq \text{LambdaGap-X} \succeq \text{LambdaGap-S}. \end{aligned}$$

The relative computational cost between LambdaRank-P@k and LambdaGap-X depends on the value of k relative to n. Specifically, LambdaGap-X is more expensive than LambdaRank-P@k when $k < (n+1)/3$. In practice, where it is common for $k \ll n$, LambdaGap-X will be more expensive than LambdaRank-P@k. Note that the number of pairwise comparisons required for LambdaGap-X represents an upper bound for the required pairwise comparisons for all LambdaGap algorithms.

Based on these observations alone, LambdaGap-S is expected to be the fastest in terms of gradient calculation per iteration. However, optimizations can also be made in the data ordering step. Most methods necessitate sorting the data because their weighting factors depend on the specific positions

of the documents in the ranking. Comparison-based sorting algorithms have a lower bound of $O(n \log n)$ average complexity. Some models, however, can avoid full sorting:

- —RankNet and BinRankNet do not require sorting because their weighting factor Δ_{ij} depends solely on scores rather than rankings.
- —LambdaRank-P@k requires only the identification of whether documents are in the top-k, not their exact order. For such cases, selection algorithms such as quickselect [11] or introselect [21] can efficiently partition the documents into top-k and non-top-k groups with an average complexity of $O(n)$.

These sorting optimizations could shift the efficiency ranking of the algorithms. For example, BinRankNet could become more efficient than LambdaRank-ARP$_b k$, given that it can avoid the full sorting step.

Importantly, pairwise comparison counts and sorting complexity alone do not fully determine computational performance of the algorithms. The binary nature of some methods reduces actual comparisons by limiting them to relevant and irrelevant document pairs (e.g., BinRankNet is expected to be more efficient than RankNet). Additionally, more complex gradient calculations may introduce overhead, as seen in algorithms such as LambdaRank-NDCG, where gain, discount, and IDCG values must be computed or retrieved.

6 Experiments

All the discussed approaches were implemented using the LightGBM library [20], which is a tree-based gradient boosting framework that supports classification, regression, and LTR models. For LTR, LightGBM uses LambdaMART [4] to optimize the NDCG. The training process begins by fitting a decision tree regressor to the relevance labels using a least-squares objective, resulting in an initial scoring model Φ_1, which maps each document to a real-valued score. In subsequent iterations, the objective function shifts to

$$l(y, s^{(t)}) = \sum_{y_i > y_j} |\Delta\text{NDCG}_{ij}| \log\left(1 + e^{-\sigma(s_i^{(t)} - s_j^{(t)})}\right)$$

averaged across all the queries, where $s^{(t)} = \Phi_t(x)$ are the scores at iteration t.

LightGBM updates the scores iteratively using second-order gradient descent. At each iteration $t > 1$, the decision tree's fitting objective is to find f_t which minimizes the second-order Taylor expansion of $l(y, s^{(t-1)} + f_t(x))$ as a function of $f_t(x)$, centered at 0. Removing constant terms, this equals

$$\sum_{i=1}^{n} \left[\frac{\partial}{\partial s_i} l(y, s^{(t-1)}) \cdot f_t(x_i) + \frac{1}{2} \frac{\partial^2}{\partial {s_i}^2} l(y, s^{(t-1)}) \cdot (f_t(x_i))^2 \right].$$

For a clear and detailed explanation of how such a tree is constructed, see [7]. The scores are then updated as $s^{(t)} = s^{(t-1)} + \alpha f_t(x)$, where α is the learning rate. The partial derivatives used in this process are as follows:

$$\frac{\partial}{\partial s_i} l(y, s) = \sum_{j: y_i > y_j} |\Delta\text{NDCG}_{ij}| \rho_{ij} - \sum_{j: y_j > y_i} |\Delta\text{NDCG}_{ji}| \rho_{ji},$$

$$\frac{\partial^2}{\partial {s_i}^2} l(y, s) = \sum_{j: y_i \neq y_j} |\Delta\text{NDCG}_{ij}| \rho_{ij} (1 - \rho_{ij}),$$

with

$$\rho_{ij} = \frac{-\sigma}{1 + e^{-\sigma(s_i - s_j)}}.$$

LightGBM also includes a parameter τ called the truncation level, which limits gradient calculations to document pairs where at least one document is within the top τ ranks, effectively implementing LambdaRank-NDCG@τ. The truncation level τ was used as the value of k when needed for LambdaRank-P@k, LambdaRank-ARP$_b k$, and LambdaGap models.

To implement other LambdaRank-like algorithms in LightGBM, one only needs to modify the weighting factor calculation. However, this does not consider the potential optimizations discussed in Section 5.5. Optimizations related to ordering data by scores can be easily implemented by adjusting the sorting method according to the chosen algorithm. However, optimizing the process by skipping unnecessary pairwise comparisons requires intelligent loop design. For instance, LambdaGap-S uses a single loop over $i = 1, 2, \ldots, n - k$, constructing document pairs from the ith and $i + k$th scoring positions, resulting in a significantly faster algorithm than LambdaMART on the NDCG. Similar optimizations were applied to other algorithms where applicable.

6.1 Datasets

We conducted our experiments using three publicly accessible datasets: Microsoft's MSLR-WEB30K [24], Istella-S LETOR [18], and Yahoo!'s LTRC [5]. All three datasets are equipped with predefined training, validation, and test splits. MSLR-WEB30K has 5 folds, whereas Istella-S LETOR and Yahoo!'s LTRC have a single split each. We utilized the first fold of MSLR-WEB30K and the first set of descriptors from Yahoo!'s dataset. The label ranges across all datasets are integers from 0 to 4 inclusive. The feature counts are 136 for MSLR-WEB30K, 220 for Istella-S LETOR, and 700 for Yahoo!'s LTRC. Notably, Yahoo!'s dataset contains missing values, constituting approximately 68% of the total values across splits. LightGBM, the base implementation of the algorithms in our experiments, natively handles these missing values. To do so, missing values are ignored when constructing the splits in the decision trees, and are assigned to the side of the split that reduces the loss the most. At inference time, missing values are always assigned to the same side of the split (either left or right).

Table 2 provides a summary of the datasets. Notably, there are significant label distribution differences among them. MSLR-WEB30K maintains a relatively balanced distribution between relevant (approximately 51%) and non-relevant (approximately 49%) documents. In contrast, a majority of Yahoo!'s dataset comprises relevant documents (approximately 74%), whereas Istella-S LETOR is overwhelmingly composed of non-relevant documents (approximately 89%).

6.2 Competing Methods

The algorithms analyzed were as follows: LambdaRank-P@k, RankNet, BinRankNet, LambdaRank-NDCG, LambdaLoss-NDCG, LambdaRank-BNDCG, LambdaLoss-BNDCG, LambdaRank-ARP$_b k$, LambdaGap-S, LambdaGap-X, LambdaGap-S+, LambdaGap-X+, LambdaGap-S++, and LambdaGap-X++. NDCG-based models are included for completeness, including LambdaLoss-NDCG, which uses the weighting factor from NDCG-Loss2 [27]:

$$\Delta_{ij} = \frac{2^{y_i} - 2^{y_j}}{\text{IDCG}} \left| \frac{1}{\log_2(|i - j| + 1)} - \frac{1}{\log_2(|i - j| + 2)} \right|.$$

The weighting factor for LambdaLoss-BNDCG is defined analogously. We also include the hybrid methods between LambdaRank and LambdaLoss applied to NDCG and BNDCG, as these methods are found to have the best performance in [27] (NDCG-Loss2++). We call these hybrid algorithms LambdaLoss-NDCG++ and LambdaLoss-BNDCG++.

Table 2. Summary of the Datasets Used in the Experiments

Dataset	Split	# Queries	# Documents	Documents per Query			Label Distribution				
				Min	Median	Max	0	1	2	3	4
WEB30K	Training	18,919	2,270,296	1	110	1,251	0.52	0.32	0.13	0.02	0.01
	Validation	6,306	747,218	1	109	679	0.51	0.32	0.13	0.02	0.01
	Test	6,306	753,611	1	109	908	0.51	0.33	0.13	0.02	0.01
Istella-S	Training	19,245	2,043,304	3	124	180	0.89	0.02	0.04	0.03	0.02
	Validation	7,211	684,076	4	94	178	0.88	0.02	0.04	0.03	0.02
	Test	6,562	681,250	3	118	182	0.89	0.03	0.04	0.03	0.02
Yahoo!	Training	19,944	473,134	1	19	139	0.26	0.36	0.28	0.08	0.02
	Validation	2,994	71,083	1	18	117	0.27	0.35	0.28	0.08	0.02
	Test	6,983	165,660	1	19	129	0.26	0.36	0.29	0.08	0.02

The label distribution is provided as the fraction of documents with each label, rounded to two decimal places.

We additionally include the Lambda-eX approach, given the potential of the Lambda-eX approach to increase the number of training signals in LambdaRank. We include all the variants discussed in Section 4.2, i.e., `static`, `random`, `all`, `all-static`, and `all-random`, and use their provided implementation.[1] These are applied to LambdaRank, LambdaLoss, and their hybrid, both with binarized and non-binarized objectives. Adopting a similar notation to [19], we refer to the algorithms as follows: LambdaMART-eX static, LambdaMART-eX random, LambdaMART-eX all, LambdaMART-eX all-static, LambdaMART-eX all-random, LambdaLoss-eX static, LambdaLoss-eX random, LambdaLoss-eX all, LambdaLoss-eX all static, LambdaLoss-eX all random, LambdaLoss-eX++ static, LambdaLoss-eX++ random, LambdaLoss-eX++ all, LambdaLoss-eX++ all static, and LambdaLoss-eX++ all random. We additionally append the "bin" suffix to the algorithms that use binarized objectives. In this case, the `all-static` and `all-random` variants can be omitted, as they always roll back to the `static` and `random` variants, respectively.

Having discussed the ARP metric, we find it appropriate to include the ARP-Loss1 and ARP-Loss2 approaches from [27], which utilize the weighting factors $\Delta_{ij} = y_i$ and $\Delta_{ij} = y_i - y_j$, respectively. These weights are applied to pairs of documents such that $y_i > y_j$, as described in Equation (1), which justifies the omission of the absolute value.

The weighting factors are derived from an alternative definition of the ARP metric proposed by the authors, expressed as $\text{ARP} = \sum_{i=1}^{n} y_i \cdot i$. This formulation incorporates graded relevance, in contrast to our binary definition of ARP. Notably, when the labels are binary, both weighting factors simplify to $\Delta_{ij} = 1$: The first represents the highest label in the pair, and the second represents the label difference between the two documents. Consequently, the binarized versions of these methods correspond to BinRankNet.

Additionally, since these weighting factors are independent of document rankings (i.e., i and j), sorting documents by relevance is unnecessary for their computation, similar to the RankNet approach. This property can lead to improved training efficiency. Hereafter, we refer to these algorithms as LambdaLoss-ARP1 and LambdaLoss-ARP2.

6.2.1 Random and Perfect Score. In addition to the proposed algorithms, we also include random and perfect models, for reference. The addition of a perfect model is useful with P@k, as the definition of the metric prevents it from taking the value of 100% if queries exist with fewer than

[1] https://github.com/FedericoMarcuzzi/LambdaRank-Gradients-are-Incoherent/.

Table 3. Hyperparameters Used for the Initial Experiment, Where the Hyperparameters Were Fixed

	WEB30K	Yahoo!	Istella-S
Learning rate	0.02	0.02	0.05
# Leaves	200	400	64
Min data in leaf	100	50	20
Min hessian	0	0	0.001

k relevant documents. The random model, on the other hand, serves as a reference model which should be beaten with a margin by any decent model.

To calculate the random score for each dataset and value of k, we use a hypergeometric distribution model. This model describes the probability of drawing m successes, or relevant documents, without replacement in k draws from a finite population of size $n \geq k$ that contains B relevant documents. The distribution is denoted as Hypergeometric(n, B, k).

Let X_q represent the random variable for the number of relevant documents retrieved from k random samples without replacement from a query q with $n_q \geq k$ documents. X_q follows a hypergeometric distribution Hypergeometric(n_q, B_q, k), where B_q is the total number of relevant documents in query q. The probability mass function is given by

$$P(X_q = m) = \frac{\binom{B_q}{m}\binom{n_q - B_q}{k-m}}{\binom{n_q}{k}},$$

where $\binom{a}{b}$ denotes the binomial coefficient. The expected value of X_q is $\mathbb{E}[X_q] = k \cdot B_q/n_q$.

Let

$$Y = \frac{1}{k|Q|}\sum_{q \in Q} X_q.$$

Then, $\mathbb{E}[Y]$ represents the expected average P@k from a random model and can be computed as

$$\mathbb{E}[Y] = \frac{1}{k|Q|}\sum_{q \in Q}\mathbb{E}[X_q] = \frac{1}{|Q|}\sum_{q \in Q}\frac{B_q}{n_q}.$$

We use this result to compute the random score for each dataset. For a given query, this result corresponds to the intuition that, on average, the proportion of relevant documents in a subset of the documents should be the same as the proportion of relevant documents overall. Note that this result does not depend on the value of k. It is easy to see that the formula still holds even if Q contains queries with $n_q < k$, as their P@k when all their documents are sampled is B_q/n_q.

6.3 Fixed Hyperparameters

To evaluate the effectiveness of the proposed approaches for the optimization of P@k, we tested each method with different values of k: 5, 10, and 15. To ensure that the observed effects of each of the methods are not due to hyperparameter selection, we employ a fixed set of hyperparameters for each dataset, which were chosen based on the literature [19] and are detailed in Table 3. This additionally allows easier comparisons between algorithms, such as being able to compare training times. The parameter μ in the hybrid LambdaGap models was fixed to the value of 1, and the truncation level τ was set to k, when applicable. All other hyperparameters were left to LightGBM's defaults.

The models were trained on the training split of each dataset for 1,000 iterations. The best iteration based on P@k on the validation set was selected and evaluated on the test set. To evaluate whether the proposed approaches represent an improvement over LambdaRank-P@k, we performed one-sided paired permutation tests that compared LambdaRank-P@k to all other methods. This type of test has the advantage of not making any assumptions on the underlying data or statistic distributions, with the downside of being much slower than other tests such as the Student's paired t-test.

To perform the paired permutation test, we first computed the P@k for all queries in each test set using two different methods A = LambdaRank-P@k and B. Let X_A and X_B denote the sets of P@k values on the test set computed using A and B, respectively, with means μ_A and μ_B. We first computed the value of $\Delta\mu_{AB} = \mu_B - \mu_A$, which represents the observed improvement in mean P@k from B over A. Next, we randomly shuffled P@k values between X_A and X_B, ensuring that we only interchanged P@k values corresponding to the same query, obtaining new sets X'_A and X'_B. We then recomputed the difference in means $\Delta\mu'_{AB}$ analogously to before. We repeated this process several times, each time obtaining different values of $\Delta\mu'_{AB}$. We finished the test by reporting a p-value, computed as the proportion of times that $\Delta\mu'_{AB} > \Delta\mu_{AB}$.

The number of times shuffling is repeated is, at most, $2^{|Q|}$, where Q is the set of queries in the test set. In practice, the use of far fewer samples is sufficient. We used 10,000 iterations of the procedure.

6.3.1 Results. The performance results for each combination of dataset, algorithm, and value of k with fixed hyperparameters are presented in Table 4. Across all datasets and configurations, the models consistently outperform the random baseline, confirming the effectiveness of the training procedure. LambdaGap-S+ and LambdaGap-X+ stand out as the only models that consistently surpass LambdaRank-P@k across all the cases studied. BinRankNet and LambdaRank-ARP$_b k$ also perform strongly, whereas LambdaRank-NDCG, LambdaLoss-NDCG, and non-binarized Lambda-eX models consistently underperform, yielding worse results than LambdaRank-P@k does. Binarized Lambda-eX models manage to beat LambdaRank-P@k in a few cases, albeit a minority, with even fewer statistically significant improvements overall.

LambdaRank's Suboptimal Performance for P@k. The results indicate that LambdaRank-P@k may not be ideal for optimizing P@k; it does not achieve the best performance in any scenario, consistently falling short against LambdaGap-S+ and LambdaGap-X+. These improvements are statistically significant in all but one case (MSLR-WEB30K for $k = 5$). Even simpler models, such as BinRankNet, outperform LambdaRank-P@k in most configurations. This underperformance is likely linked to the design of the P@k metric, which treats all documents outside the top k as irrelevant, making it difficult for LambdaRank-P@k push documents beyond the top k towards the top k.

Competitive Performance of LambdaRank-ARP$_b k$. LambdaRank-ARP$_b k$ outperforms LambdaRank-P@k in all but one case (Istella-S with $k = 15$), with most improvements being statistically significant. Notably, it achieves the best performance for MSLR-WEB30K at $k = 5$. This suggests that ARP$_b k$ may be particularly effective for P@k optimization, given that it disregards the order within the top k, while pushing documents outside the top k more strongly than in LambdaRank-P@k.

Binarization Enhances Performance. A consistent theme across datasets is the superior performance of binarized algorithms, such as BinRankNet, LambdaRank-BNDCG, LambdaLoss-BNDCG, and binarized versions of Lambda-eX compared with their non-binary counterparts. This trend highlights the effectiveness of binarized weighting factors when optimizing for P@k, even though

Table 4. Performance Comparison of Various Ranking Algorithms with No Hyperparameter Tuning, Measured by P@k (%)

	MSLR-WEB30K Fold 1			Istella-S			Yahoo! Set 1		
	$k=5$	$k=10$	$k=15$	$k=5$	$k=10$	$k=15$	$k=5$	$k=10$	$k=15$
Perfect model	94.56	91.93	89.53	97.07	89.27	75.59	92.13	87.68	85.09
Random model	44.97	44.97	44.97	12.81	12.81	12.81	75.30	75.30	75.30
LambdaRank-P@k	73.52	69.32	66.32	91.30	81.70	68.85	84.22	81.82	80.42
RankNet	73.32	69.10	66.35	91.17	81.54	68.86	84.23	81.79	80.37
BinRankNet	73.30	69.24	66.35	92.12***	**82.31*****	**69.39*****	84.67***	**82.17*****	**80.63*****
LambdaRank-NDCG	72.06	68.08	65.19	88.31	79.04	67.22	83.70	81.31	79.88
LambdaLoss-NDCG	70.74	66.49	63.73	87.69	78.48	66.98	83.50	81.15	79.79
LambdaLoss-NDCG++	72.08	68.10	65.44	88.38	79.27	67.61	83.83	81.40	80.00
LambdaRank-BNDCG	73.47	69.27	66.33	91.24	81.47	68.74	84.19	81.90	80.44
LambdaLoss-BNDCG	73.68	68.92	65.26	91.47*	82.09***	69.13***	84.40*	82.09***	80.55**
LambdaLoss-BNDCG++	73.53	69.40	66.21	91.22	81.62	68.84	84.25	81.92*	80.49*
LambdaRank-ARP$_b$$k$	**73.88***	69.52*	66.46	91.76***	81.75	68.83	84.70***	82.13***	80.52**
LambdaLoss-ARP1	73.12	68.98	66.05	90.78	81.26	68.64	84.01	81.54	80.16
LambdaLoss-ARP2	73.18	69.24	66.38	91.22	81.67	68.94*	84.02	81.58	80.28
LambdaMART-eX static	71.40	67.81	65.27	88.70	79.49	67.49	83.53	81.38	79.92
LambdaMART-eX random	71.25	67.84	65.34	88.76	79.48	67.57	83.70	81.35	79.92
LambdaMART-eX all	71.54	68.03	65.37	88.84	79.52	67.53	83.84	81.32	79.95
LambdaMART-eX all static	71.67	67.98	65.34	88.84	79.57	67.45	83.74	81.38	79.97
LambdaMART-eX all random	71.58	67.95	65.34	88.87	79.56	67.54	83.78	81.40	79.97
LambdaLoss-eX static	67.83	65.11	62.96	87.63	79.22	67.38	83.07	81.01	79.76
LambdaLoss-eX random	67.22	64.96	63.08	87.78	79.34	67.51	83.13	80.96	79.78
LambdaLoss-eX all	67.99	65.72	63.43	87.94	79.40	67.49	83.19	81.09	79.74
LambdaLoss-eX all static	67.87	65.45	63.42	87.80	79.37	67.53	83.29	81.00	79.81
LambdaLoss-eX all random	67.98	65.48	63.44	88.01	79.30	67.50	83.06	81.06	79.75
LambdaLoss-eX++ static	71.02	67.65	65.10	88.96	80.16	68.13	83.69	81.42	79.99
LambdaLoss-eX++ random	71.04	67.58	65.05	89.00	80.12	68.19	83.62	81.41	80.04
LambdaLoss-eX++ all	71.23	67.84	65.34	89.15	80.26	68.14	83.72	81.40	80.07
LambdaLoss-eX++ all static	71.28	67.90	65.22	89.19	80.15	68.19	83.56	81.46	80.05
LambdaLoss-eX++ all random	71.08	67.84	65.31	89.21	80.27	68.17	83.60	81.46	80.02
LambdaMART-eX static bin	73.39	69.26	66.30	91.30	81.61	68.80	84.03	81.80	80.41
LambdaMART-eX random bin	73.48	69.29	66.28	91.26	81.67	68.82	84.10	81.78	80.43
LambdaMART-eX all bin	73.58	69.28	66.41	91.30	81.66	68.82	84.18	81.88	80.33
LambdaLoss-eX static bin	73.40	68.86	64.91	91.29	81.80*	68.77	84.25	81.75	80.44
LambdaLoss-eX random bin	72.88	68.32	64.95	91.32	81.72	68.78	84.29	81.83	80.41
LambdaLoss-eX all bin	71.92	68.03	65.24	91.20	81.82*	68.84	84.30	81.93*	80.42
LambdaLoss-eX++ static bin	73.41	69.31	66.31	91.30	81.61	68.79	84.17	81.75	80.41
LambdaLoss-eX++ random bin	73.49	69.24	66.29	91.24	81.63	68.78	84.22	81.74	80.34
LambdaLoss-eX++ all bin	72.96	69.12	66.26	91.36	81.62	69.01*	84.15	81.92*	80.39
LambdaGap-S	71.58	67.89	65.83	**92.20*****	81.94***	68.76	84.10	81.98**	80.44
LambdaGap-X	73.77	69.72***	66.58**	91.77***	81.57	68.50	84.62***	82.04***	80.39
LambdaGap-S+	73.53	**69.76*****	66.59***	91.96***	81.90***	68.92*	84.56***	82.09***	80.52**
LambdaGap-X+	73.85*	69.70***	**66.62*****	91.89***	81.91***	68.91*	**84.81*****	82.11***	80.51**
LambdaGap-S++	73.87*	69.65**	66.43	91.68***	81.76	68.84	84.80***	82.09***	80.56***
LambdaGap-X++	73.71	69.63**	66.42	91.72***	81.76	68.83	84.59***	82.12***	80.51*

Asterisks indicate statistically significant improvement with respect to LambdaRank-P@k, determined through one-sided paired permutation tests (*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$). The best performance in each column, excluding the perfect model, is highlighted in bold. The algorithms that perform worse than LambdaRank-P@k are highlighted in red.

some information loss may occur. Binarized models prioritize pushing a larger number of low-relevance documents to the top ranks, as opposed to fewer highly relevant documents, which is beneficial for P@k optimization.

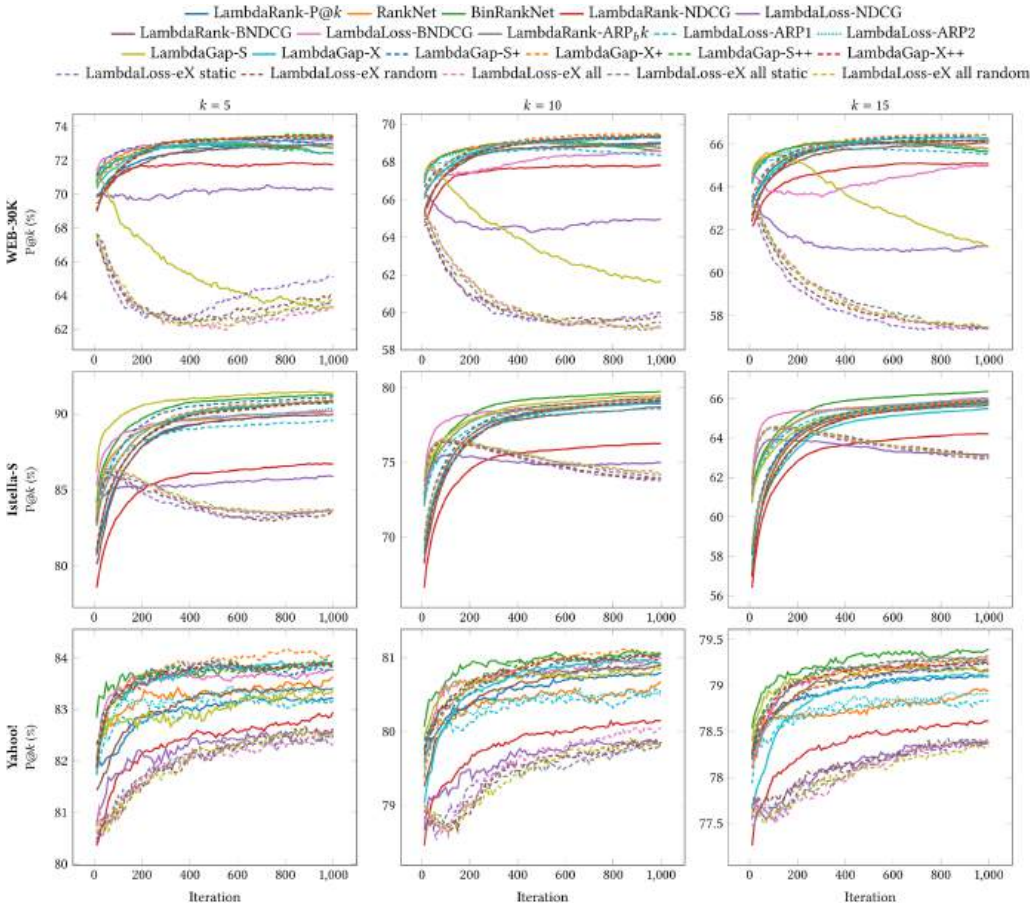

Fig. 3. Progression of P@k across training iterations on the validation set, shown for all combinations of algorithm, dataset, and value of k. The first 10 iterations, where performance is markedly lower, are excluded to reduce visual clutter.

Figures 3 and 4 illustrate the evolution of P@k across training iterations for both the training and validation datasets. To improve interpretability, we only include the non-binarized, non-hybrid LambdaLoss-eX variants among the Lambda-eX models, as they display the most distinctive trends. Hybrid LambdaLoss variants are similarly excluded, as their behavior closely aligns with other included models, making them redundant for this particular analysis.

Model Overfitting and Generalization. The presented figures reveal significant overfitting patterns, particularly among certain LambdaLoss-based and LambdaGap-based models. Notably, LambdaGap-S and non-binarized LambdaLoss variants, including LambdaLoss-eX, demonstrate a tendency to overfit, achieving high performance on the training set but exhibiting difficulties in generalizing to validation data. This overfitting effect is especially severe on the MSLR-WEB30K dataset, where the validation performance declines sharply as training progresses. In the Istella-S dataset, a similar trend is observed, albeit to a lesser degree.

Conversely, the LambdaGap-S+ and LambdaGap-S++ models maintain a more stable balance between training and validation performance, suggesting that they generalize more effectively.

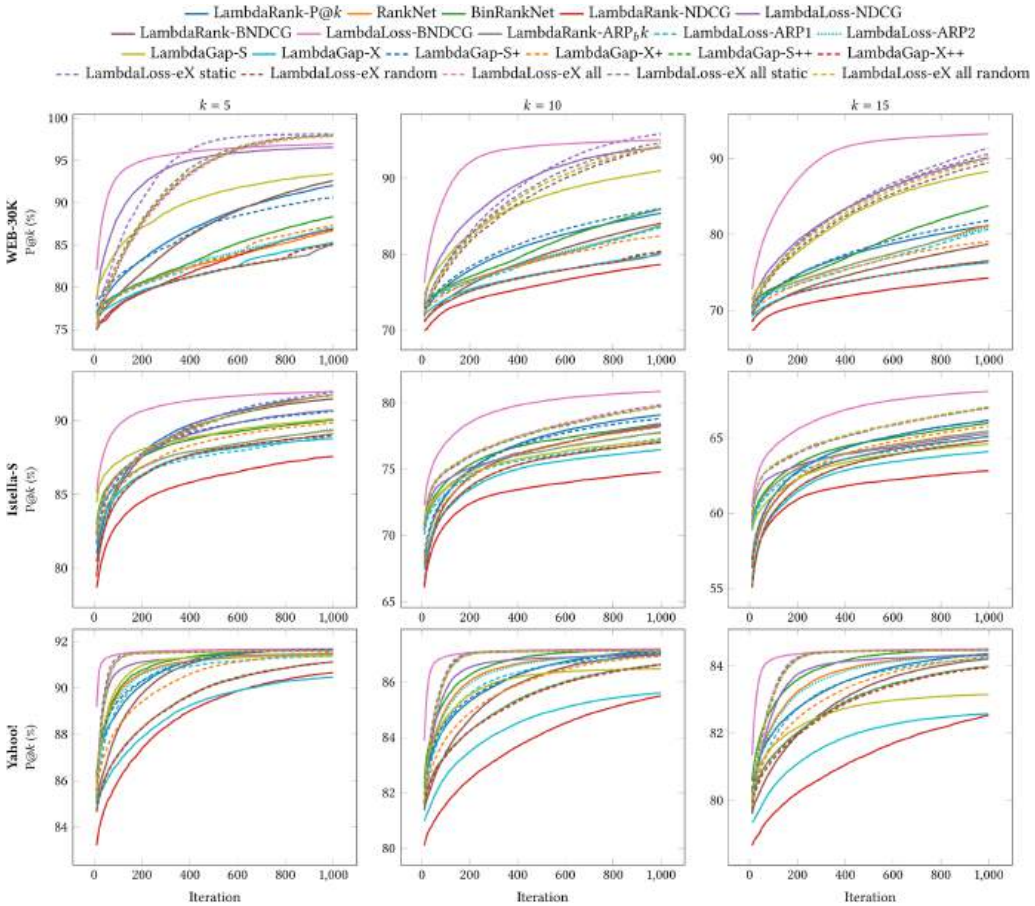

Fig. 4. Progression of P@k across training iterations on the training set, shown for all combinations of algorithm, dataset, and value of k. The first 10 iterations, where performance is markedly lower, are excluded to reduce visual clutter.

This finding indicates that hybrid LambdaGap formulations, particularly when integrated with LambdaRank, act as a form of implicit regularization, reducing overfitting and leading to more robust performance across datasets. A similar regularization effect is evident in the binarized models, as reflected in the contrast between LambdaLoss-NDCG and its binarized counterpart, LambdaLoss-BNDCG. These results underscore the importance of selecting an appropriate loss function variant to mitigate overfitting while optimizing ranking precision.

Poor Performance of NDCG-Based Models. NDCG-based algorithms, including LambdaRank-NDCG, LambdaLoss-NDCG, and non-binarized Lambda-eX variants, consistently exhibit suboptimal performance across all evaluation settings. This underperformance is likely due to a fundamental misalignment between their optimization objectives and the P@k metric. Since these algorithms prioritize ranking documents by their relevance scores rather than directly maximizing the number of relevant documents in the top k positions, they fail to effectively optimize P@k.

Figures 3 and 4 further illustrate this issue, showing that LambdaRank-NDCG, despite being a common default for LTR implementations, struggles to improve P@k even during training.

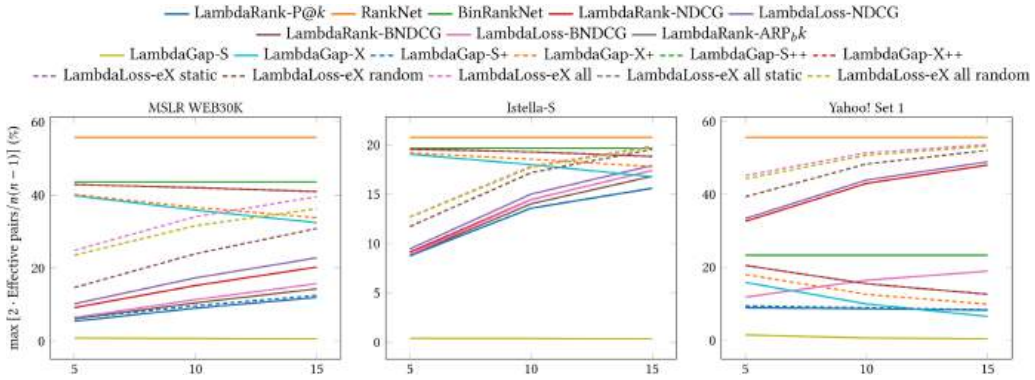

Fig. 5. Maximum number of effective pairs across different values of k for each algorithm and dataset combination. The values are normalized relative to the theoretical maximum, $n(n-1)/2$, and averaged over all queries. LambdaGap-S++ is not visible because it overlaps with LambdaRank-ARP$_b k$, although their values are not exactly identical. A similar overlap occurs between LambdaRank-BNDCG and LambdaLoss-BNDCG in the Yahoo! dataset.

Given these findings, LambdaRank-NDCG should be avoided for tasks where P@k is the primary evaluation metric, and alternative approaches that explicitly target precision-based ranking should be considered.

Dataset-Specific Performance Variability. No single ranking algorithm consistently outperforms all others across different datasets and values of k. Although the LambdaGap models generally deliver strong results, their advantage is not universal. Notably, BinRankNet stands out as the top-performing model in four of the nine scenarios analyzed, particularly excelling on the Istella-S and Yahoo! datasets for $k = 10$ and $k = 15$. Our observations suggest that LambdaGap models may be more effective at lower values of k, where the emphasis is on optimizing the top-ranked documents. In contrast, BinRankNet seems better suited for larger cutoff values. Testing this hypothesis with larger values of k (e.g., $k = 100$) would offer a more comprehensive understanding of these algorithms' performance across a broader range of cutoff thresholds. However, our primary focus remains on the lower cutoff values, which are most critical for ranking tasks.

6.3.2 Training Times. As already noted, the number of required pairwise comparisons for each algorithm may significantly impact its training efficiency. During training, pairs for which the weighting factor is non-zero are termed *effective pairs*. In Section 5.5, we provide proxies for the expected number of effective pairs, which are dependent on the algorithm and the value of k relative to n. However, the actual number of effective pairs is also influenced by the labels of the documents, as only pairs with different labels are compared. For binary algorithms, such as BinRankNet, only pairs where exactly one document is relevant are compared, which can lead to a much lower number of comparisons than the theoretical maximum of $n(n-1)/2$.

Figure 5 displays the maximum number of effective pairs for each algorithm and dataset as a function of k. In the interest of visualization clarity, several algorithms are omitted, because they largely or even perfectly overlap with other algorithms. In particular, LambdaLoss-ARP1 and LambdaLoss-ARP2 are skipped because they are guaranteed to match RankNet.

The number of effective pairs stabilizes rapidly across all algorithms (see Appendix Section A.2; Figure A1), indicating the efficiency of the training process once model convergence begins. An exception is Lambda-eX, where the number of effective pairs starts high and decreases as training

progresses. This occurs because, early in training, many *missed top-k* documents prompt Lambda-eX to expand the full-gradient set. As training converges and the number of *missed top-k* documents declines, the number of effective pairs aligns with those of non-Lambda-eX models.

The relative rankings of the algorithms, as discussed in Section 5.5, remain largely consistent. RankNet consistently shows the highest number of effective pairs, whereas LambdaGap-S has the lowest. BinRankNet ranks next in MSLR-WEB30K and Istella-S, but on Yahoo!, it is surpassed by NDCG-based models. This discrepancy is due to the high number of relevant documents in the Yahoo! dataset with varying relevance labels, which are better utilized by non-binary models such as RankNet, LambdaRank-NDCG, and LambdaLoss-NDCG. This also explains the larger gap between non-binarized models and their binarized counterparts, with datasets featuring more relevant documents exhibiting the widest gaps.

LambdaRank-ARP$_b k$, LambdaGap-X, and their hybrid variants closely follow BinRankNet and exhibit a declining trend as k increases, as expected. In contrast, the more efficient LambdaRank-P@k and LambdaGap-S+ show an increasing trend with k, narrowing the gap with models such as LambdaGap-X, except for Yahoo!. This exception can be attributed to the smaller median query size of 19 in Yahoo!'s training split, which affects the relative impact of k in relation to n. Nevertheless, the rapid decline in the number of effective pairs for LambdaGap-X with respect to k renders it more efficient than LambdaRank-P@k on Yahoo! when $k = 15$.

As shown in Table 5, the total training times for each algorithm-dataset combination and value of k reveal that LambdaGap-S is generally the fastest algorithm across all datasets and values of k. This aligns with the expectation that fewer effective pairs lead to faster training. RankNet, despite having the highest number of effective pairs, does not always exhibit the slowest training times, likely because it can avoid sorting documents.

LambdaRank-ARP$_b k$ and LambdaGap-X, which also involve a large number of pairwise comparisons, are slower, particularly when k is small compared with n. This is evident in the MSLR-WEB30K and Istella-S datasets, where the median query sizes are relatively large (110 and 124, respectively). In contrast, for the Yahoo! dataset, where the median query size is much smaller, the training times for several models become faster than those of LambdaRank-P@k. This suggests that the relationship between effective pairs and training times is not straightforward and can be influenced by other factors, such as the complexity of gradient calculations and the specific optimizations implemented by each algorithm.

Lambda-eX holds the promise of providing similar computational efficiency to the algorithms on which it is based, sometimes even exhibiting superior performance. This is likely due to implementation details, rather than algorithmic differences.

To better understand the implications of these training times, Table 6 presents the training times until the best validation iteration for each combination of algorithm, dataset, and value of k. This metric can be a more practical indicator of the required training time, as it reflects the effectiveness of early stopping. Interestingly, the LambdaGap-S and LambdaLoss models (including some Lambda-eX variants) often find their best validation iteration significantly faster than other models do, likely because of their propensity to overfit quickly, and the fast per-iteration training times for LambdaGap-S due to its lower requirement for pairwise comparisons, as discussed earlier. Conversely, for other models, the time to the best iteration generally correlates with the total training time, as the best iteration in the validation set comes close to the end of the training process.

6.4 Hyperparameter Optimization

In a second training phase, we search for the optimal hyperparameters for each combination of dataset, algorithm, and value of k. This allows a more faithful comparison in terms of performance

Table 5. Training Times for Each Algorithm-Dataset Combination and Value of k, Expressed as a Percentage Relative to the Training Time of LambdaRank-P@k

	MSLR-WEB30K Fold 1			Istella-S			Yahoo! Set 1		
	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$
LambdaRank-P@k	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
RankNet	147.70	145.29	143.25	119.02	110.97	109.64	78.08	81.53	78.20
BinRankNet	139.00	134.86	132.92	116.95	110.50	109.10	83.47	83.80	81.72
LambdaRank-NDCG	101.89	104.03	101.91	89.53	98.05	104.91	91.75	93.18	91.58
LambdaLoss-NDCG	79.66	77.96	78.55	82.07	84.62	88.63	80.60	79.96	77.83
LambdaLoss-NDCG++	93.02	93.72	94.82	88.18	97.25	101.82	91.16	92.96	84.92
LambdaRank-BNDCG	100.91	104.59	103.89	102.92	100.53	103.68	98.99	101.56	93.98
LambdaLoss-BNDCG	91.93	88.75	78.85	101.09	96.85	96.25	89.73	87.99	81.98
LambdaLoss-BNDCG++	96.70	96.44	96.11	96.39	97.33	100.36	96.19	99.92	92.38
LambdaRank-ARP$_b k$	172.61	150.72	143.94	128.16	127.14	125.83	86.12	88.05	82.75
LambdaLoss-ARP1	149.16	145.36	141.02	116.09	113.09	109.28	79.42	84.44	79.21
LambdaLoss-ARP2	150.85	147.43	142.38	115.04	113.14	111.17	79.42	83.60	79.37
LambdaMART-eX static	98.28	101.93	107.16	93.59	101.90	106.17	87.25	91.37	84.53
LambdaMART-eX random	97.53	104.49	108.85	96.93	104.71	109.20	89.57	92.38	85.29
LambdaMART-eX all	102.27	112.11	118.45	96.60	101.80	106.92	88.05	90.92	84.09
LambdaMART-eX all static	101.70	110.99	115.62	95.24	103.33	106.19	88.77	89.43	84.38
LambdaMART-eX all random	104.53	113.91	117.87	98.67	107.93	181.11	89.02	90.83	85.04
LambdaLoss-eX static	67.54	72.85	77.91	79.07	85.92	89.93	74.57	77.64	71.41
LambdaLoss-eX random	75.77	79.45	82.43	84.59	89.38	92.54	77.97	79.31	72.63
LambdaLoss-eX all	71.14	81.75	87.41	80.13	86.06	89.77	75.67	77.49	71.72
LambdaLoss-eX all static	70.27	80.37	85.86	79.77	86.72	90.19	76.39	76.20	68.99
LambdaLoss-eX all random	72.61	83.46	88.39	82.80	90.71	134.06	77.36	75.51	71.55
LambdaLoss-eX++ static	88.00	95.52	100.32	93.34	101.74	106.48	87.78	88.09	80.44
LambdaLoss-eX++ random	92.25	97.70	102.94	97.78	103.92	108.78	89.69	89.43	80.99
LambdaLoss-eX++ all	95.35	105.71	112.68	95.40	101.65	106.94	88.47	87.66	79.90
LambdaLoss-eX++ all static	94.73	105.39	109.97	95.59	103.82	105.15	87.83	85.99	79.71
LambdaLoss-eX++ all random	96.30	108.24	112.40	98.59	105.77	140.76	89.27	87.00	80.25
LambdaMART-eX static bin	111.64	101.38	101.74	93.11	99.09	104.84	95.85	103.45	92.83
LambdaMART-eX random bin	104.82	100.17	103.78	100.62	102.84	107.93	96.71	99.98	93.59
LambdaMART-eX all bin	118.44	135.00	141.44	95.73	101.44	107.82	107.27	101.06	92.65
LambdaLoss-eX static bin	92.48	80.48	77.25	94.26	90.88	93.74	91.03	87.25	78.72
LambdaLoss-eX random bin	100.84	87.06	83.94	99.45	97.53	97.04	91.07	89.22	81.01
LambdaLoss-eX all bin	87.74	94.94	94.01	93.57	91.39	94.01	96.32	87.30	79.17
LambdaLoss-eX++ static bin	107.66	98.97	100.51	92.84	98.11	101.57	96.06	99.32	91.53
LambdaLoss-eX++ random bin	100.29	101.28	103.31	99.69	104.57	108.05	101.83	99.56	93.35
LambdaLoss-eX++ all bin	113.23	132.38	137.60	94.50	100.17	103.45	104.36	99.00	92.30
LambdaGap-S	57.65	58.91	61.37	74.37	87.08	92.39	80.85	84.06	88.09
LambdaGap-X	141.80	165.86	134.03	121.55	121.17	120.88	81.84	97.64	99.21
LambdaGap-S+	141.48	140.22	137.79	106.99	116.91	117.99	80.66	94.75	96.56
LambdaGap-X+	148.45	155.50	149.43	126.14	125.36	122.23	83.64	97.76	100.82
LambdaGap-S++	153.38	155.38	148.45	130.44	125.92	129.30	85.62	87.47	85.39
LambdaGap-X++	152.05	153.73	148.71	133.45	127.86	127.39	85.61	88.17	85.18

The values reported represent the geometric mean of 10 runs. All computations were conducted on an 8-core, 3.6 GHz Intel Core i7-4790 CPU computer with 32 GB of memory, running the GNU/Linux OS with Python 3.11.6.

than when the same set of hyperparameters is used for all the cases, as some hyperparameters may work better for some models than for others. To accomplish this, we leverage the automated hyperparameter tuning library FLAML [26].

Table 6. Training Times until the Selected Iteration for Each Algorithm-Dataset Combination and Value of k, Expressed as a Percentage Relative to the Training Time of LambdaRank-P@k

	MSLR-WEB30K Fold 1			Istella-S			Yahoo! Set 1		
	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$	$k = 5$	$k = 10$	$k = 15$
LambdaRank-P@k	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
RankNet	102.79	80.47	66.85	117.00	110.97	107.15	93.03	81.59	74.71
BinRankNet	94.34	74.96	67.98	115.66	110.51	109.00	75.13	74.91	72.66
LambdaRank-NDCG	88.93	105.25	74.25	92.75	95.04	101.11	132.96	91.64	90.84
LambdaLoss-NDCG	59.71	2.53	2.80	83.41	15.13	13.43	105.24	75.24	71.20
LambdaLoss-NDCG++	45.31	83.97	94.84	88.75	93.78	101.37	131.24	92.91	83.80
LambdaRank-BNDCG	101.88	105.94	102.01	106.49	100.55	103.58	133.35	98.55	80.34
LambdaLoss-BNDCG	89.32	86.62	76.56	102.57	96.33	93.71	127.74	86.52	81.20
LambdaLoss-BNDCG++	83.28	97.14	95.10	99.22	96.69	100.11	139.23	97.39	91.75
LambdaRank-ARP$_b k$	165.06	151.27	140.12	132.36	126.30	125.11	122.37	85.62	81.20
LambdaLoss-ARP1	72.58	92.43	61.29	119.67	113.00	108.73	77.32	82.50	62.04
LambdaLoss-ARP2	112.30	73.34	70.47	117.28	113.14	108.55	69.36	57.88	68.04
LambdaMART-eX static	87.33	101.50	103.68	96.86	101.12	105.71	88.15	89.76	84.44
LambdaMART-eX random	94.49	96.95	103.64	97.86	104.76	109.25	129.83	90.28	85.49
LambdaMART-eX all	98.54	98.69	107.17	95.97	101.02	106.38	117.09	64.62	79.05
LambdaMART-eX all static	86.22	104.97	89.61	98.45	102.95	106.24	124.13	83.01	80.99
LambdaMART-eX all random	94.45	112.62	116.50	97.52	104.75	179.96	114.16	90.54	84.66
LambdaLoss-eX static	2.45	1.99	2.34	10.39	12.80	16.53	88.05	77.03	71.30
LambdaLoss-eX random	2.10	2.63	2.93	9.25	11.40	16.03	101.11	77.71	71.76
LambdaLoss-eX all	2.54	3.36	3.13	9.33	12.06	17.00	104.39	77.36	69.07
LambdaLoss-eX all static	2.15	3.14	2.63	9.43	13.02	12.39	106.84	68.44	65.23
LambdaLoss-eX all random	2.14	2.52	3.57	10.58	13.37	33.73	94.52	66.84	68.61
LambdaLoss-eX++ static	70.01	80.47	76.46	92.74	100.91	106.44	86.64	84.91	78.14
LambdaLoss-eX++ random	48.08	80.14	85.35	95.89	103.05	107.55	76.90	87.53	81.09
LambdaLoss-eX++ all	49.39	73.22	110.20	94.67	101.49	106.70	94.38	85.59	78.12
LambdaLoss-eX++ all static	90.19	93.15	104.09	98.68	103.86	104.92	88.19	78.85	64.01
LambdaLoss-eX++ all random	48.23	105.47	99.48	101.26	105.82	140.75	88.16	65.94	67.48
LambdaMART-eX static bin	111.70	92.53	100.65	96.57	97.58	104.74	123.10	97.56	92.23
LambdaMART-eX random bin	105.22	100.39	99.08	104.16	100.46	105.92	87.35	88.11	77.03
LambdaMART-eX all bin	113.50	114.90	137.91	98.93	96.93	106.88	105.81	99.29	90.34
LambdaLoss-eX static bin	93.58	82.37	2.61	89.82	89.96	93.50	108.29	75.99	62.65
LambdaLoss-eX random bin	53.30	87.40	3.18	102.45	97.16	97.08	95.17	76.89	69.24
LambdaLoss-eX all bin	5.26	3.86	4.41	93.71	91.43	94.05	121.46	78.23	67.24
LambdaLoss-eX++ static bin	105.16	94.50	93.05	93.29	98.16	98.58	132.99	98.35	80.63
LambdaLoss-eX++ random bin	100.42	93.24	102.93	101.21	103.14	107.93	137.81	68.59	75.79
LambdaLoss-eX++ all bin	113.53	135.55	123.45	97.29	100.14	101.46	140.91	99.01	76.66
LambdaGap-S	2.05	4.62	8.13	71.28	86.19	92.36	110.81	82.53	62.91
LambdaGap-X	140.52	169.59	127.61	125.38	120.23	119.44	74.71	91.75	90.07
LambdaGap-S+	88.70	142.72	137.23	110.05	116.35	117.89	53.33	89.75	96.08
LambdaGap-X+	130.93	152.27	149.42	127.27	125.07	122.19	103.77	86.16	90.27
LambdaGap-S++	135.19	159.11	112.10	135.18	125.48	129.29	120.81	84.88	79.90
LambdaGap-X++	128.43	145.39	144.00	138.39	127.85	127.42	76.86	81.35	64.47

The values reported represent the geometric mean of 10 runs. All computations were conducted on an 8-core, 3.6 GHz Intel Core i7-4790 CPU computer with 32 GB of memory, running the GNU/Linux OS with Python 3.11.6.

FLAML uses an algorithm known as **Cost-Frugal Optimization (CFO)**, which is built upon FLOW^2. Hyperparameter tuning with FLOW^2 starts by evaluating an initial set of hyperparameters, denoted as p, through model training on a training dataset and evaluation on a separate validation

Table 7. Hyperparameters Considered for the Hyperparameter Search with FLAML

Hyperparameter	Distribution	Range
Number of leaves	lograndint	$[2^4 - 1, 2^8 - 1]$
Truncation level	choice	$[k, k + 3]$
Min child weight	uniform	$[0, 1]$
Learning rate	loguniform	$[1/1,024, 1.0]$
Min child samples	lograndint	$[2, 2^7 + 1]$

set. Each hyperparameter configuration is represented as a d-dimensional real-valued vector, where each component corresponds to a specific hyperparameter.

The tuning process involves sampling a random vector u from the d-dimensional unit sphere, scaling it by a fixed factor δ, and generating a new candidate set of hyperparameters, $p^+ = p + \delta u$. This new candidate is evaluated in the same manner as p. If p^+ outperforms p, it becomes the new reference point. Otherwise, a second candidate $p^- = p - \delta u$ is evaluated. If p^- also fails to improve performance, the original set p is retained. This iterative refinement process continues for a predefined number of iterations.

CFO enhances FLOW^2 by incorporating practical adjustments, such as projecting the candidate hyperparameters p^+ and p^- onto a feasible hyperparameter space, which is crucial when hyperparameters have discrete values. Additionally, CFO includes randomized restarts to avoid local optima by reverting to the initial hyperparameter set with a small random perturbation. For a comprehensive explanation of the CFO algorithm, we refer readers to [26].

To use the FLAML library, we selected an initial set of hyperparameters drawn from Table 3, and specified a distribution for each hyperparameter from which to sample. The distributions used are shown in Table 7. A truncation level of $\tau = k + 3$ is included, as suggested in [20]. An additional truncation level $\tau = 1,000$ was added for LambdaRank-NDCG, LambdaRank-BNDCG, LambdaLoss-NDCG, and LambdaLoss-BNDCG, which effectively results in no truncation. In the LambdaGap hybrid models, μ was sampled from a uniform distribution of $[0.1, 10]$. The number of iterations was set to 40. The same sampling strategy was used for the hybrid LambdaLoss models. For the Lambda-eX models, we treat the choice of strategy for constructing the full-gradient set as a hyperparameter, randomly sampled from `static`, `random`, `all`, `all-static`, and `all-random`, while we continue to separate binarized and non-binarized versions, which we call *Lambda-eX bin* and *Lambda-eX*, respectively.

The models were trained using the respective training splits and selected based on their P@k performance on the corresponding validation splits. Finally, the chosen models were evaluated on their respective test sets, including the statistical testing outlined in Section 6.3.1. The outcomes of this hyperparameter tuning process are presented in Table 8, and the selected hyperparameters are listed in Appendix Section A.1, Tables A1–A3.

The results in Table 8 significantly outperform those in Table 4, where hyperparameter tuning was not applied. The newly introduced LambdaRank-ARP$_b k$ and LambdaGap models consistently exceeded LambdaRank-P@k in most scenarios, demonstrating their superior effectiveness in optimizing P@k.

Improved Effectiveness of LambdaRank-P@k. Hyperparameter tuning led to substantial performance gains for LambdaRank-P@k, significantly improving its results across all datasets and rank cutoffs compared with its untuned counterpart. In particular, the tuned version of LambdaRank-P@k now outperforms other models in a greater number of cases than before, suggesting that it can be

Table 8. Performance Comparison of Various Ranking Algorithms with Hyperparameter Tuning, Measured by P@k (%)

	MSLR-WEB30K Fold 1			Istella-S			Yahoo! Set 1		
	$k=5$	$k=10$	$k=15$	$k=5$	$k=10$	$k=15$	$k=5$	$k=10$	$k=15$
Perfect model	94.56	91.93	89.53	97.07	89.27	75.59	92.13	87.68	85.09
Random model	44.97	44.97	44.97	12.81	12.81	12.81	75.30	75.30	75.30
LambdaRank-P@k	73.98	69.77	66.62	92.41	83.06	69.80	84.41	81.92	80.44
RankNet	73.32	69.23	66.36	91.19	81.93	69.21	84.25	81.81	80.39
BinRankNet	73.31	69.48	66.41	92.63**	83.47***	70.27***	84.69**	82.19***	80.65***
LambdaRank-NDCG	72.61	68.48	65.66	90.09	80.72	68.43	83.72	81.33	79.89
LambdaLoss-NDCG	72.11	68.08	65.27	89.29	80.36	68.44	83.52	81.16	79.82
LambdaLoss-NDCG++	72.60	68.51	65.77	89.46	80.46	68.40	83.86	81.42	80.03
LambdaRank-BNDCG	73.88	69.59	66.43	92.54*	83.28***	70.13***	84.39	81.93	80.46*
LambdaLoss-BNDCG	73.71	69.38	66.22	92.49	83.26***	70.10***	84.41	82.10**	80.57***
LambdaLoss-BNDCG++	73.94	69.72	66.37	92.52	83.07	70.14***	84.29	81.93	80.51*
LambdaRank-ARP$_b$$k$	73.90	69.53	66.48	92.62**	83.26***	69.77	84.72***	82.14***	80.53**
LambdaLoss-ARP1	73.13	69.04	66.13	90.80	81.72	69.04	84.02	81.64	80.27
LambdaLoss-ARP2	73.26	69.27	66.40	91.24	82.14	69.51	84.16	81.68	80.39
Lambda-eX	72.24	68.26	65.45	89.68	80.73	68.78	83.85	81.56	80.20
Lambda-eX bin	73.87	69.70	66.67	92.48	82.98	69.25	84.44	82.04*	80.55**
LambdaGap-S	72.82	68.75	66.14	92.86***	82.32	68.78	84.53	81.99*	80.45*
LambdaGap-X	73.85	69.73	66.61	92.85***	82.50	68.85	84.66**	82.14***	80.51**
LambdaGap-S+	**74.17**	70.01**	**66.94*****	92.81***	83.35***	70.26***	84.57*	82.19***	80.63***
LambdaGap-X+	74.15	**70.11*****	66.84**	92.86***	83.41***	70.26***	**84.83*****	82.21***	80.64***
LambdaGap-S++	73.97	69.69	66.59	92.81***	83.41***	70.26***	84.82***	82.20***	**80.68*****
LambdaGap-X++	73.83	69.76	66.50	**92.91*****	**83.49*****	**70.29*****	84.62*	**82.21*****	80.63***

Asterisks indicate statistically significant improvement with respect to LambdaRank-P@k, determined through one-sided paired permutation tests (*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$). The best performance in each column, excluding the perfect model, is highlighted in bold. The algorithms that perform worse than LambdaRank-P@k are highlighted in red.

effective for P@k optimization when its hyperparameters are carefully selected. However, despite these improvements, LambdaRank-P@k still falls short of the LambdaGap models, indicating that it may possess inherent limitations in optimizing P@k as effectively. The model's sensitivity to hyperparameter choices further suggests that it requires extensive tuning to reach competitive performance, making it less robust than LambdaGap across different datasets.

LambdaGap Achieves Top Performance. Among all the evaluated models, LambdaGap-X++ emerged as the top performer, achieving the highest scores in four out of nine test cases. Across datasets, the hybrid LambdaGap variants LambdaGap-S++, LambdaGap-X++, LambdaGap-S+, and LambdaGap-X+—exhibited statistically significant improvements over LambdaRank-P@k in nearly all settings. The only exception was MSLR-WEB30K at $k = 5$, where no model achieved statistical significance. These results reinforce the strong potential of the LambdaGap models for P@k optimization, as they consistently outperform RankNet, BinRankNet, and all NDCG-based algorithms.

Binarization Remains Important. The consistently poor performance of the RankNet, LambdaRank-NDCG, LambdaLoss-NDCG, and non-binarized Lambda-eX models relative to LambdaRank-P@k confirms that the weaknesses of these algorithms are not simply due to inadequate hyperparameter

tuning. However, their binarized counterparts BinRankNet, LambdaRank-BNDCG, LambdaLoss-BNDCG, and binarized Lambda-eX demonstrated notable improvements in a few cases, particularly on the Istella-S and Yahoo! datasets. These results reinforce the conclusion that binarization is generally beneficial for P@k optimization, as it helps models focus on maximizing the number of possibly lower-relevance documents in the top ranks rather than fewer more relevant ones.

BinRankNet Performs Competitively. BinRankNet continued to perform well on the Istella-S and Yahoo! datasets, closely approaching the best-performing models at $k = 10$ and $k = 15$. Although LambdaGap-S++ and LambdaGap-X++ achieved slightly higher scores at these rank cutoffs, the differences were not statistically significant ($p > 0.05$). Given that BinRankNet requires significantly less training time than LambdaGap models while delivering comparable performance in certain cases, it represents a practical option for large-scale ranking tasks where training resources are constrained.

6.5 Effect of μ

The LambdaGap hybrid models introduced in Section 5.4 include a new hyperparameter, μ (Equation (2)), which controls the contribution of each model in the combined weighting factor of the resulting LambdaRank-like algorithms. Lower values of μ (close to 0) emphasize the influence of either LambdaRank-P@k or LambdaRank-ARP$_b k$, whereas higher values increase the effect of LambdaGap-S or LambdaGap-X. When μ is set to 1, both models contribute equally.

In Section 6.4, we tuned μ for various models and found that 25 out of the 36 selected LambdaGap hybrid models performed best with $\mu > 1$. This suggests that, in most cases, the contribution of the LambdaGap models outweighs that of the LambdaRank models.

To further examine the sensitivity of the LambdaGap models to μ, we retrained each model using the optimized hyperparameters from Section 6.4, while varying μ in the range $[0.1, 10]$ in steps of 1.1. As before, we selected the best iteration based on the validation performance and evaluated the final model on the test set.

Figure 6 shows the dependence of test performance on μ. Contrary to the findings in [27], we do not observe a consistent trend where higher μ values lead to superior performance. In most cases, the performance remains stable across the studied range, indicating that μ has a limited effect on performance improvements within this range. This suggests that simply counting how often $\mu > 1$ does not adequately reflect the importance of each combined model.

For instance, on the Yahoo! dataset with $k = 5$, the best performance for LambdaGap-X+ is achieved with $\mu = 10$, yet the performance at $\mu = 0.1$ is nearly identical. Conversely, for Istella-S with $k = 5$ and LambdaGap-X++, the opposite pattern emerges. These results highlight that while μ influences model performance, its impact is less straightforward than initially anticipated.

When comparing the performance of hybrid models against their respective LambdaRank and LambdaGap baselines, most hybrid configurations surpass the baseline performance. For some datasets, such as Istella-S with $k = 5$ or Yahoo! with $k = 10$, the extent of improvement does depend on the value of μ. Thus, although the performance remains relatively stable across μ, fine-tuning this hyperparameter is crucial for surpassing the LambdaRank and LambdaGap baselines in some cases.

In summary, while μ provides a mechanism for balancing the contributions of different models in hybrid algorithms, its overall effect is less pronounced than expected. Nonetheless, careful tuning of μ can still lead to incremental improvements over baseline models, suggesting that μ remains a relevant factor in specific scenarios.

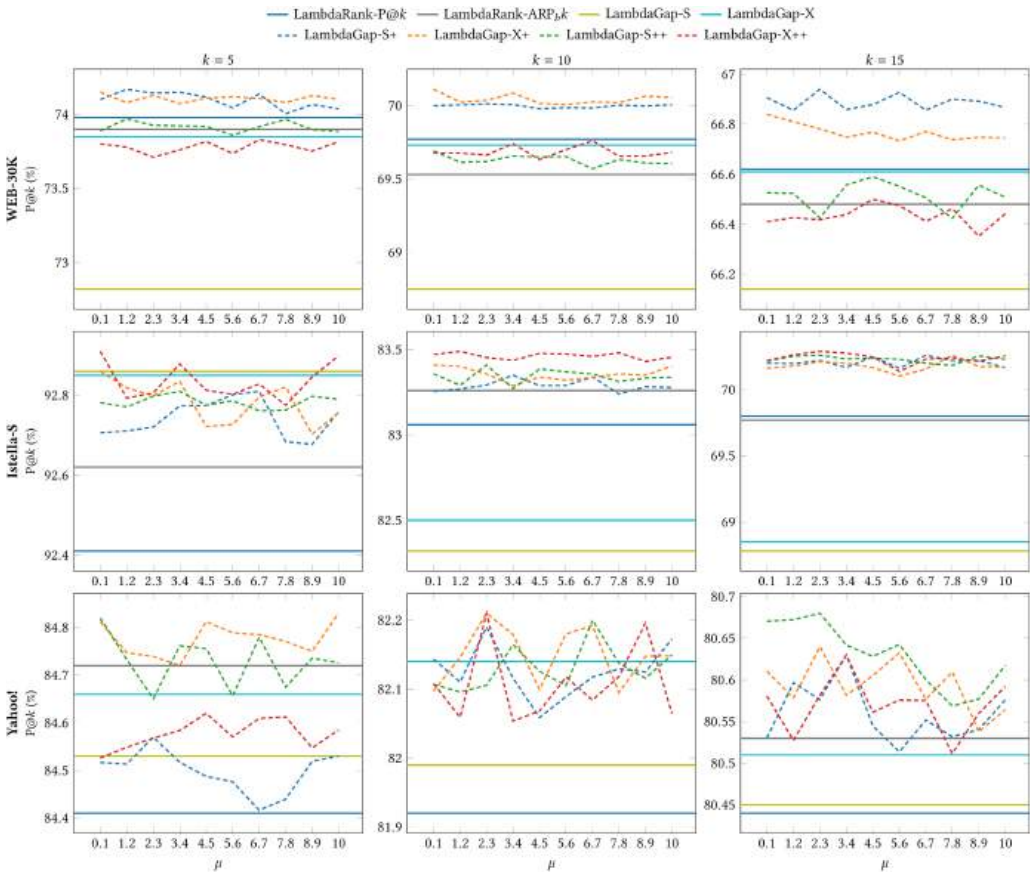

Fig. 6. Test performance of the LambdaGap hybrid models with different values of μ. The baseline performance of the individual models used to construct the hybrid algorithms is shown as horizontal lines.

7 Conclusion

In this work, we introduced several alternatives to LambdaRank for optimizing P@k, including the proposed LambdaGap framework. We demonstrated that LambdaGap directly optimizes a metric that is closely aligned with P@k while incorporating a penalization mechanism for the presence of non-top-k relevant documents, thus offering a more nuanced optimization criterion. Additionally, we introduced a novel metric, ARP$_b k$, designed to overcome some of the inherent limitations associated with direct P@k optimization using LambdaRank. We further showed that employing binarized versions of existing ranking algorithms can enhance P@k optimization by prioritizing the push of a greater number of low-relevance documents towards top ranks. Moreover, we proposed integrating the LambdaGap models into hybrid frameworks that combine the strengths of LambdaRank and LambdaGap, which could offer a robust solution for P@k optimization. In particular, we showed that LambdaGap-S+ provided direct optimization of P@k, while utilizing strictly more document pairs than LambdaRank-P@k does, including some outside the top-k.

Our empirical evaluation on three publicly available LTR datasets substantiated the effectiveness of the proposed approaches, revealing that several alternatives consistently outperform the traditional LambdaRank-P@k. In particular, the LambdaGap-based hybrid models emerged as top

performers, demonstrating statistically significant improvements over other models in most scenarios, both with and without extensive hyperparameter tuning. LambdaRank-ARP$_b k$ and BinRankNet (the binarized version of RankNet) also achieved strong performance, indicating their suitability for P@k optimization tasks. In contrast, models primarily designed for NDCG optimization yielded suboptimal results for P@k optimization, highlighting the importance of selecting optimization frameworks that are directly aligned with the target metric.

Future Work. This work paves the way for future research on P@k optimization by establishing a theoretical foundation for LambdaGap. With a clear understanding of the underlying metric optimized by LambdaGap, future research can explore alternative weighting schemes to further refine the LambdaGap approach and potentially mitigate overfitting.

While our analysis shows that LambdaGap-S maximizes a lower bound for P@k, an open question remains regarding the tightness of this bound. Investigating whether the gap between this bound and the true P@k is significant could lead to a more refined theoretical understanding of LambdaGap's optimization properties.

Additionally, while LambdaGap has been validated on large-scale LTR benchmarks, its effectiveness in real-world retrieval systems where P@k is the primary objective remains unexplored. Future research could integrate LambdaGap into production search or recommendation pipelines to assess its practical benefits, such as in [1, 10].

Finally, LambdaGap is designed for ranking metrics where the ordering within the top k is irrelevant, primarily P@k. While Recall@k appears to be the only other widespread metric sharing this property, further work could explore whether other metrics exist that benefit from the LambdaGap framework. A formal analysis of the relationship between P@k and Recall@k in this context may also provide additional theoretical insights.

References

[1] Ramon Adàlia, Shivani Patel, Anthony Paiva, Tierni Kaufman, Ismael Zamora, Xianmei Cai, Gemma Sanjuan, and Wilson Z. Shou. 2024. Development of a predictive multiple reaction monitoring (MRM) model for high-throughput ADME analyses using learning-to-rank (LTR) techniques. *Journal of the American Society for Mass Spectrometry* 35, 1 (Nov. 2023), 131–139. DOI: https://doi.org/10.1021/jasms.3c00363

[2] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. In *Proceedings of the Advances in Neural Information Processing Systems*. B. Schölkopf, J. Platt, and T. Hoffman (Eds.), Vol. 19, MIT Press. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf

[3] 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)*. ACM, New York, NY, 89–96. DOI: https://doi.org/10.1145/1102351.1102363

[4] Chris J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report MSRTR2010-82. Retrieved from https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

[5] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge (Proceedings of Machine Learning Research)*, Vol. 14. Olivier Chapelle, Yi Chang, and Tie-Yan Liu (Eds.), PMLR, Haifa, Israel, 1–24. Retrieved from https://proceedings.mlr.press/v14/chapelle11a.html

[6] Olivier Chapelle and Mingrui Wu. 2010. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval* 13, 3 (2010), 216–235. DOI: https://doi.org/10.1007/s10791-009-9110-3

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, 785–794. DOI: https://doi.org/10.1145/2939672.2939785

[8] Pinar Donmez, Krysta M. Svore, and Christopher J. C. Burges. 2009. On the local optimality of LambdaRank. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '09)*. ACM, New York, NY, 460–467. DOI: https://doi.org/10.1145/1571941.1572021

[9] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4 (Dec. 2003), 933–969.

[10] Deisy Morselli Gysi, Ítalo do Valle, Marinka Zitnik, Asher Ameli, Xiao Gan, Onur Varol, Susan Dina Ghiassian, J. J. Patten, Robert A. Davey, Joseph Loscalzo, et al. 2021. Network medicine framework for identifying drug-repurposing opportunities for COVID-19. *Proceedings of the National Academy of Sciences* 118, 19 (2021), e2025581118. DOI: https://doi.org/10.1073/pnas.2025581118

[11] Charles Anthony Richard Hoare. 1961. Algorithm 65: Find. *Communications of the ACM* 4, 7 (Jul. 1961), 321–322. DOI: https://doi.org/10.1145/366622.366647

[12] Rolf Jagerman, Zhen Qin, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2022. On optimizing top-K metrics for neural ranking models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. ACM, New York, NY, 2303–2307. DOI: https://doi.org/10.1145/3477495.3531849

[13] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '00)*. ACM, New York, NY, 41–48. DOI: https://doi.org/10.1145/345508.345545

[14] Xin Jiang, Yunhua Hu, and Hang Li. 2009. A ranking approach to keyphrase extraction. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '09)*. ACM, New York, NY, 756–757. DOI: https://doi.org/10.1145/1571941.1572113

[15] Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*. ACM, New York, NY, 377–384. DOI: https://doi.org/10.1145/1102351.1102399

[16] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, 781–789. DOI: https://doi.org/10.1145/3018661.3018699

[17] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331. DOI: https://doi.org/10.1561/1500000016

[18] 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)*. ACM, New York, NY, 949–952. DOI: https://doi.org/10.1145/2911451.2914763

[19] Federico Marcuzzi, Claudio Lucchese, and Salvatore Orlando. 2023. LambdaRank gradients are incoherent. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23)*. ACM, New York, NY, 1777–1786. DOI: https://doi.org/10.1145/3583780.3614948

[20] Microsoft. 2023. LightGBM Documentation: Parameters. Retrieved from https://lightgbm.readthedocs.io/en/stable/Parameters.html

[21] David R. Musser. 1997. Introspective sorting and selection algorithms. *Software: Practice and Experience* 27, 8 (1997), 983–993. Retrieved from https://onlinelibrary.wiley.com/doi/10.1002/%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23

[22] Yashoteja Prabhu and Manik Varma. 2014. FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, 263–272. DOI: https://doi.org/10.1145/2623330.2623651

[23] Ronak Pradeep, Rodrigo Frassetto Nogueira, and Jimmy Lin. 2021. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. arXiv:2101.05667. Retrieved from https://arxiv.org/abs/2101.05667

[24] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv:*1306.2597. Retrieved from http://arxiv.org/abs/1306.2597

[25] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397. DOI: https://doi.org/10.1007/s10791-009-9124-x

[26] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A fast and lightweight AutoML library. In *Proceedings of Machine Learning and Systems 3 (MLSys)*.

[27] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The LambdaLoss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. ACM, New York, NY, 1313–1322. DOI: https://doi.org/10.1145/3269206.3271784

[28] Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: Learning to rank with joint word-image embeddings. *Machine Learning* 81, 1 (Oct. 2010), 21–35. DOI: https://doi.org/10.1007/s10994-010-5198-3

[29] Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (Jun. 2010), 254–270. DOI: https://doi.org/10.1007/s10791-009-9112-1

[30] Biao Xiang, Daxin Jiang, Jian Pei, Xiaohui Sun, Enhong Chen, and Hang Li. 2010. Context-aware ranking in web search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '10)*. ACM, New York, NY, 451–458. DOI: https://doi.org/10.1145/1835449.1835525

A Appendix

A.1 Hyperparameter Tuning Results

Table A1. Hyperparameter Tuning Results for MSLR-WEB30K, Rounded to Four Decimal Places

k	Algorithm	Num. Leaves	Min Samples	Learning Rate	τ	Min Weight	μ
5	LambdaRank-P@k	175	116	0.0481	8	0.0171	-
5	RankNet	65	93	0.0494	5	0.0000	-
5	BinRankNet	254	116	0.0273	8	0.0010	-
5	LambdaRank-NDCG	131	30	0.0269	1,000	0.0002	-
5	LambdaLoss-NDCG	41	17	0.0013	8	0.0000	-
5	LambdaLoss-NDCG++	200	100	0.0200	8	0.0017	1.7714
5	LambdaRank-BNDCG	159	128	0.0387	8	0.0553	-
5	LambdaLoss-BNDCG	204	106	0.0082	5	0.0000	-
5	LambdaLoss-BNDCG++	122	42	0.0571	8	0.0048	2.4147
5	LambdaRank-ARP$_b k$	116	86	0.0412	5	0.0091	-
5	LambdaLoss-ARP1	233	104	0.0132	8	0.0002	-
5	LambdaLoss-ARP2	212	87	0.0096	8	0.0005	-
5	Lambda-eX	197	95	0.0452	8	0.0013	0.1000
5	Lambda-eX bin	150	65	0.0339	8	0.0031	0.3092
5	LambdaGap-S	32	3	0.0020	5	0.0903	-
5	LambdaGap-X	162	40	0.0212	8	0.0150	-
5	LambdaGap-S+	169	13	0.0597	8	0.0631	0.6687
5	LambdaGap-X+	212	114	0.0237	8	0.0035	0.1000
5	LambdaGap-S++	254	59	0.0335	5	0.0174	0.8352
5	LambdaGap-X++	254	57	0.0265	8	0.0079	7.2016
10	LambdaRank-P@k	146	118	0.0526	13	0.0183	-
10	RankNet	217	128	0.0184	13	0.0135	-
10	BinRankNet	145	101	0.0100	13	0.0002	-
10	LambdaRank-NDCG	254	122	0.0481	1,000	0.0081	-
10	LambdaLoss-NDCG	93	25	0.0013	13	0.0000	-
10	LambdaLoss-NDCG++	182	128	0.0320	13	0.0124	2.1776
10	LambdaRank-BNDCG	247	128	0.0627	13	0.0185	-
10	LambdaLoss-BNDCG	68	30	0.0010	13	0.0000	-
10	LambdaLoss-BNDCG++	143	128	0.0373	13	0.0030	1.6795
10	LambdaRank-ARP$_b k$	202	67	0.0344	13	0.0012	-
10	LambdaLoss-ARP1	254	115	0.0097	13	0.0006	-
10	LambdaLoss-ARP2	254	116	0.0097	13	0.0006	-
10	Lambda-eX	180	84	0.0349	13	0.0029	0.1000
10	Lambda-eX bin	233	114	0.0466	13	0.0012	0.1000
10	LambdaGap-S	32	2	0.0230	13	0.0424	-
10	LambdaGap-X	163	85	0.0267	13	0.0000	-
10	LambdaGap-S+	121	113	0.0344	13	0.0009	2.6088
10	LambdaGap-X+	213	128	0.0320	13	0.0033	0.1000
10	LambdaGap-S++	151	63	0.0511	10	0.0143	0.1000
10	LambdaGap-X++	247	21	0.0320	10	0.0857	6.5094
15	LambdaRank-P@k	226	127	0.0574	18	0.0174	-
15	RankNet	204	106	0.0082	18	0.0006	-
15	BinRankNet	254	116	0.0097	18	0.0006	-
15	LambdaRank-NDCG	254	84	0.0251	1,000	0.0237	-
15	LambdaLoss-NDCG	51	30	0.0010	18	0.0000	-
15	LambdaLoss-NDCG++	254	32	0.0192	18	0.0030	2.3570
15	LambdaRank-BNDCG	116	86	0.0412	18	0.0049	-
15	LambdaLoss-BNDCG	69	23	0.0010	18	0.0000	-
15	LambdaLoss-BNDCG++	254	67	0.0743	18	0.0051	0.4182
15	LambdaRank-ARP$_b k$	251	75	0.0303	15	0.0355	-
15	LambdaLoss-ARP1	254	65	0.0085	18	0.0008	-
15	LambdaLoss-ARP2	207	128	0.0129	18	0.0007	-
15	Lambda-eX	190	94	0.0284	18	0.0044	0.1000
15	Lambda-eX bin	190	94	0.0284	18	0.0044	0.1000
15	LambdaGap-S	23	13	0.0469	15	0.0228	-
15	LambdaGap-X	202	93	0.0298	18	0.0000	-
15	LambdaGap-S+	250	128	0.0530	18	0.0157	1.8465
15	LambdaGap-X+	165	128	0.0550	18	0.0017	0.1000
15	LambdaGap-S++	240	128	0.0381	15	0.0459	4.6494
15	LambdaGap-X++	107	87	0.0684	15	0.0570	4.8387

Table A2. Hyperparameter Tuning Results for Istella-S, Rounded to Four Decimal Places

k	Algorithm	Num. Leaves	Min Samples	Learning Rate	τ	Min Weight	μ
5	LambdaRank-P@k	164	26	0.1400	8	0.0097	-
5	RankNet	140	32	0.0762	8	0.0738	-
5	BinRankNet	233	30	0.2166	8	0.0114	-
5	LambdaRank-NDCG	221	33	0.1443	1,000	0.0194	-
5	LambdaLoss-NDCG	154	24	0.0107	8	0.0032	-
5	LambdaLoss-NDCG++	250	42	0.0378	8	0.0075	1.9091
5	LambdaRank-BNDCG	164	26	0.1400	8	0.0097	-
5	LambdaLoss-BNDCG	254	30	0.1055	8	0.0145	-
5	LambdaLoss-BNDCG++	213	11	0.1880	8	0.0018	0.1000
5	LambdaRank-ARP$_b k$	215	15	0.1595	8	0.0813	-
5	LambdaLoss-ARP1	107	25	0.0450	8	0.0036	-
5	LambdaLoss-ARP2	62	10	0.0921	8	0.0018	-
5	Lambda-eX	84	29	0.0668	8	0.0004	0.7908
5	Lambda-eX bin	140	19	0.1401	8	0.0000	0.1000
5	LambdaGap-S	254	9	0.0650	5	0.0634	-
5	LambdaGap-X	254	43	0.1526	5	0.0835	-
5	LambdaGap-S+	246	18	0.0779	5	0.0910	7.1534
5	LambdaGap-X+	254	2	0.3130	5	0.0114	0.5066
5	LambdaGap-S++	218	11	0.2754	8	0.0925	3.6084
5	LambdaGap-X++	254	2	0.2849	5	0.0216	0.1000
10	LambdaRank-P@k	233	36	0.1720	13	0.0197	-
10	RankNet	107	25	0.0450	13	0.0036	-
10	BinRankNet	235	25	0.0954	13	0.0517	-
10	LambdaRank-NDCG	254	36	0.1172	1,000	0.1000	-
10	LambdaLoss-NDCG	91	18	0.0039	13	0.0002	-
10	LambdaLoss-NDCG++	250	42	0.0378	13	0.0075	1.9091
10	LambdaRank-BNDCG	254	36	0.1720	13	0.0197	-
10	LambdaLoss-BNDCG	254	65	0.0297	13	0.0060	-
10	LambdaLoss-BNDCG++	219	6	0.0801	13	0.0130	0.8352
10	LambdaRank-ARP$_b k$	252	2	0.1563	10	0.0714	-
10	LambdaLoss-ARP1	132	20	0.0447	13	0.0009	-
10	LambdaLoss-ARP2	98	23	0.0612	13	0.0012	-
10	Lambda-eX	84	29	0.0668	13	0.0004	0.7908
10	Lambda-eX bin	121	31	0.0592	13	0.0001	1.4529
10	LambdaGap-S	165	128	0.0585	10	0.0610	-
10	LambdaGap-X	158	25	0.0679	10	0.0706	-
10	LambdaGap-S+	254	2	0.1696	10	0.0265	3.1473
10	LambdaGap-X+	242	59	0.1540	10	0.0781	0.3772
10	LambdaGap-S++	254	2	0.3985	10	0.0209	2.0169
10	LambdaGap-X++	254	3	0.1991	10	0.0208	0.7133
15	LambdaRank-P@k	254	64	0.1314	18	0.1000	-
15	RankNet	98	23	0.0612	18	0.0012	-
15	BinRankNet	254	33	0.0634	18	0.0241	-
15	LambdaRank-NDCG	254	33	0.1593	1,000	0.0346	-
15	LambdaLoss-NDCG	117	13	0.0018	18	0.0001	-
15	LambdaLoss-NDCG++	250	42	0.0378	18	0.0075	1.9091
15	LambdaRank-BNDCG	254	29	0.2589	18	0.0421	-
15	LambdaLoss-BNDCG	170	29	0.0233	18	0.0011	-
15	LambdaLoss-BNDCG++	254	67	0.1601	18	0.0169	0.1000
15	LambdaRank-ARP$_b k$	171	96	0.3269	18	0.0045	-
15	LambdaLoss-ARP1	76	16	0.0741	18	0.0058	-
15	LambdaLoss-ARP2	98	23	0.0612	18	0.0012	-
15	Lambda-eX	124	65	0.0568	18	0.0000	1.3977
15	Lambda-eX bin	98	32	0.0956	18	0.0004	0.1000
15	LambdaGap-S	97	128	0.0763	15	0.0310	-
15	LambdaGap-X	72	6	0.0669	15	0.0523	-
15	LambdaGap-S+	247	16	0.1876	15	0.0765	6.2260
15	LambdaGap-X+	218	17	0.2102	15	0.0279	7.6016
15	LambdaGap-S++	254	2	0.3985	15	0.0209	2.0169
15	LambdaGap-X++	254	2	0.3572	15	0.0149	1.9260

Table A3. Hyperparameter Tuning Results for Yahoo!, Rounded to Four Decimal Places

k	Algorithm	Num. Leaves	Min Samples	Learning Rate	τ	Min Weight	μ
5	LambdaRank-P@k	19	47	0.0491	5	0.0002	-
5	RankNet	15	43	0.0412	5	0.0002	-
5	BinRankNet	36	43	0.0596	5	0.0001	-
5	LambdaRank-NDCG	27	52	0.0361	8	0.0006	-
5	LambdaLoss-NDCG	19	53	0.0082	5	0.0000	-
5	LambdaLoss-NDCG++	15	67	0.1052	5	0.0000	2.3668
5	LambdaRank-BNDCG	33	80	0.0995	5	0.0008	-
5	LambdaLoss-BNDCG	15	39	0.0064	5	0.0000	-
5	LambdaLoss-BNDCG++	18	42	0.1114	5	0.0001	1.8476
5	LambdaRank-ARP$_b$$k$	181	31	0.0126	8	0.0821	-
5	LambdaLoss-ARP1	26	72	0.0098	5	0.0001	-
5	LambdaLoss-ARP2	27	44	0.0137	5	0.0001	-
5	Lambda-eX	19	50	0.0200	5	0.0001	1.0000
5	Lambda-eX bin	19	47	0.0452	5	0.0000	0.1000
5	LambdaGap-S	98	2	0.0132	8	0.0701	-
5	LambdaGap-X	40	71	0.0747	5	0.0007	-
5	LambdaGap-S+	30	39	0.0318	5	0.0001	2.6816
5	LambdaGap-X+	249	7	0.0255	8	0.0535	10.0000
5	LambdaGap-S++	254	33	0.0295	5	0.0464	0.4858
5	LambdaGap-X++	254	22	0.0351	8	0.0721	3.9551
10	LambdaRank-P@k	33	79	0.1236	10	0.0004	-
10	RankNet	21	112	0.0231	10	0.0002	-
10	BinRankNet	19	47	0.0491	10	0.0002	-
10	LambdaRank-NDCG	24	65	0.0627	10	0.0007	-
10	LambdaLoss-NDCG	16	34	0.0019	10	0.0000	-
10	LambdaLoss-NDCG++	30	39	0.0318	10	0.0001	2.6816
10	LambdaRank-BNDCG	87	65	0.1738	10	0.0018	-
10	LambdaLoss-BNDCG	18	28	0.0024	10	0.0000	-
10	LambdaLoss-BNDCG++	66	34	0.0472	10	0.0018	1.3333
10	LambdaRank-ARP$_b$$k$	230	8	0.0148	10	0.0532	-
10	LambdaLoss-ARP1	19	71	0.0835	10	0.0008	-
10	LambdaLoss-ARP2	29	49	0.0503	10	0.0001	-
10	Lambda-eX	16	88	0.0429	10	0.0001	1.3422
10	Lambda-eX bin	20	49	0.1097	10	0.0000	0.1000
10	LambdaGap-S	220	31	0.0367	10	0.0559	-
10	LambdaGap-X	27	88	0.1648	10	0.0004	-
10	LambdaGap-S+	132	2	0.0789	10	0.0225	2.0608
10	LambdaGap-X+	113	3	0.0760	10	0.0197	2.0176
10	LambdaGap-S++	74	124	0.0404	10	0.0780	6.2634
10	LambdaGap-X++	41	128	0.0463	10	0.0004	2.4385
15	LambdaRank-P@k	52	51	0.1470	15	0.0003	-
15	RankNet	68	40	0.0118	15	0.0005	-
15	BinRankNet	60	37	0.0559	15	0.0005	-
15	LambdaRank-NDCG	58	73	0.0509	1,000	0.0038	-
15	LambdaLoss-NDCG	15	51	0.0224	15	0.0001	-
15	LambdaLoss-NDCG++	25	26	0.0180	15	0.0000	3.9510
15	LambdaRank-BNDCG	31	67	0.1795	15	0.0014	-
15	LambdaLoss-BNDCG	18	31	0.0012	15	0.0000	-
15	LambdaLoss-BNDCG++	18	44	0.0546	15	0.0000	2.7476
15	LambdaRank-ARP$_b$$k$	85	6	0.0669	18	0.0584	-
15	LambdaLoss-ARP1	15	52	0.0654	15	0.0002	-
15	LambdaLoss-ARP2	24	102	0.0214	15	0.0003	-
15	Lambda-eX	22	60	0.0629	15	0.0000	0.9726
15	Lambda-eX bin	23	128	0.0372	15	0.0000	0.1725
15	LambdaGap-S	26	58	0.0780	15	0.0002	-
15	LambdaGap-X	84	128	0.0526	15	0.0100	-
15	LambdaGap-S+	135	15	0.0359	15	0.0020	2.9512
15	LambdaGap-X+	62	30	0.1289	15	0.0007	2.5618
15	LambdaGap-S++	18	44	0.0546	15	0.0000	2.7476
15	LambdaGap-X++	32	31	0.0995	15	0.0000	3.8372

A.2 Effective Pairs

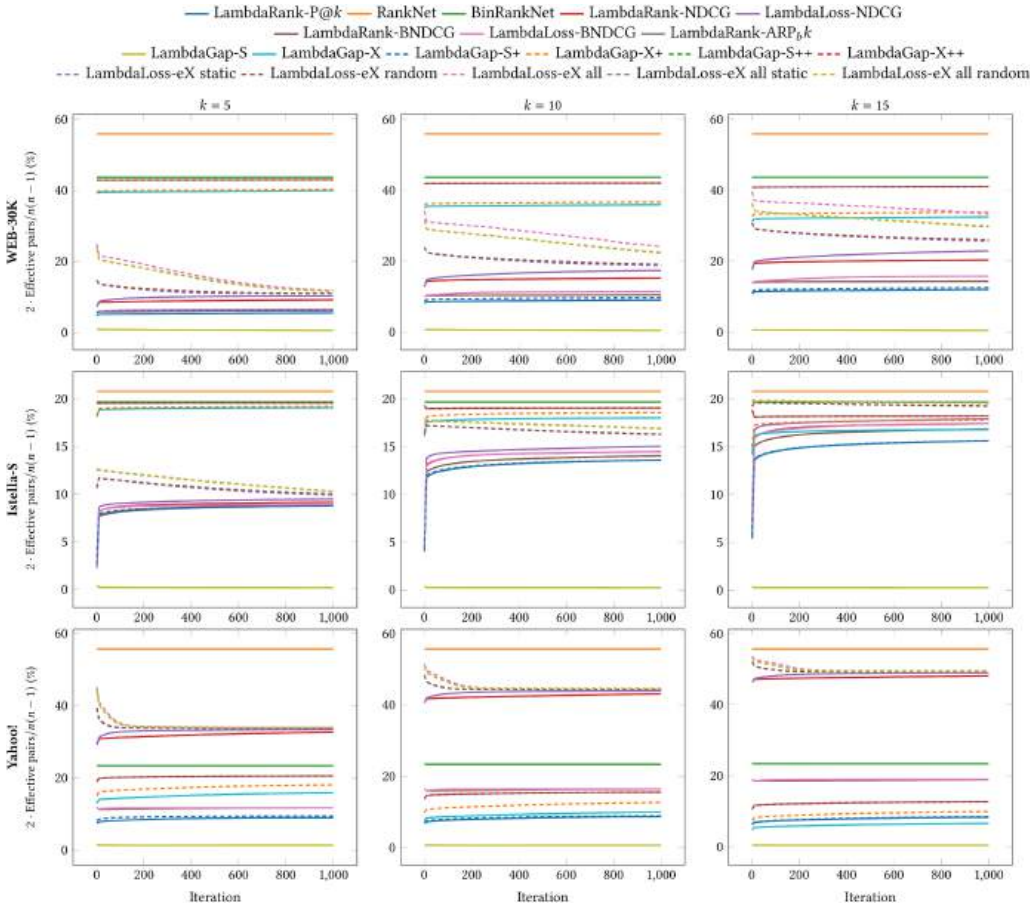

Fig. A1. Progression of effective pairs across training iterations, shown for all combinations of algorithm, dataset, and value of k. The values are expressed relative to the theoretical maximum $n(n-1)/2$, and averaged across all queries.

Received 30 October 2024; revised 26 February 2025; accepted 14 April 2025