Radboud University

Faculty of Science

A Fast and Accurate Intervention-Aware Estimator

Leveraging Counterfactual Estimation for Online Learning-to-Rank

Thesis MSc Computing Science

Supervisor:
Dr. Harrie Oosterhuis

Author:
Jingwei Kang

Second reader:
Prof. dr. ir. Arjen P. de Vries

August 2023

Abstract

As a core component of modern information retrieval (IR), ranking systems aim to present users with a sorted list of documents that match their query. Learning to rank, a research direction that employs supervised machine learning techniques, optimizes a ranking model by combining hundreds of ranking features. However, traditional learning to rank (LTR) approaches that rely on expert annotations for training face significant challenges, such as the difficulty of obtaining accurate relevance labels.

Researchers have proposed using user interaction data, such as click data, as an alternative to expert annotations for training Unbiased Learning to Rank (ULTR) models. These models include counterfactual and online LTR and aim to learn unbiased ranking models from potentially biased user interaction data. Recent developments in this field, such as the ability to combine online and counterfactual LTR in a single method, present exciting opportunities for further research and innovation.

This thesis critically analyzes the computational efficiency of previous state-of-the-art intervention-aware estimators which unify online and counterfactual LTR. The main goal of this research is to contribute to the development of efficient and effective methods for counterfactual bias estimation and model training. Since the existing method is impractical due to its time-consuming nature, this research explores a range of strategies to reduce runtime without compromising performance compared to the state-of-the-art intervention-aware method. The goal is for faster methods to perform more interventions and thus be more efficient.

By evaluating a number of different approaches, such as speeding up bias estimation using Plackett-Luce ranking models and using iterative training instead of training ranking model from scratch, this thesis aims to advance our understanding of intervention-aware estimator and its potential applications in real-world IR systems.

Contents

1 INTRODUCTION

1.1 Ranking Systems and Applications

In recent decades, the Internet has experienced explosive growth in the amount of information available. This has made efficient and effective information retrieval (IR) more important than ever, and IR systems have become essential for applications that depend on them to deliver accurate and relevant results to users [37].

Ranking is the core task of modern information retrieval and aims to present the user with a sorted list of documents that match their query. Formally, given a query q and a set of documents D matching this query, the ranking system must order the documents $d \in D$ by some criterion, such that the most relevant ones appear at the top of the list displayed to the user. Ranking systems have been used in a variety of applications, including search engines and recommendation systems. For example, search engines such as Google, Bing, etc. use ranking algorithms to determine the relevance of web pages to a user's explicit query [15]. Similarly, recommendation systems like those used by Amazon, Netflix, and YouTube use ranking algorithms to recommend products or content that match a user's interests, which are equivalent to using implicit queries to search. These systems are designed to provide users with the most relevant and useful information or recommendations based on their individual preferences and behavior.

Over the years, many ranking models have been proposed and used in the IR field. Traditional ranking models include Boolean Retrieval Models [28], Vector Space Models [27], Probabilistic Ranking Models [30, 31] and PageRank [2], etc. As the amount of web data grows, current web search engines are no longer satisfied with using a single ranking model. To improve the accuracy of retrieving the most relevant search results, researchers naturally think of studying how to combine these ranking features and create more effective new models. Building ranking models with hundreds of ranking features using supervised machine learning techniques, known as learning to rank (LTR) [13], has become a well-established research direction in the field of IR over the past two decades due to machine learning's demonstrated ability to combine multiple features.

In the LTR field, we think of the ranking system as a scoring function that assigns a score to each candidate document based on a given query and the ranking features of the document. The objective is to learn an optimal scoring function so that the most relevant documents are ranked first. There are several metrics that are commonly used to evaluate whether this objective has been achieved, such as Mean reciprocal rank (MRR), Mean average precision (MAP) and Discounted Cumulative Gain (DCG). The problem of learning to rank is often defined as an optimization problem for one of these metrics.

However, traditional learning to rank approaches that use supervised learning methods for training face certain challenges. One such challenge is obtaining accurate relevance labels: Traditional LTR methods rely on expert annotations as the judgment of relevance, that is, experts have to annotate the relevance of each query-document pair [29]. There are some limitations with annotated datasets: (i) Creating them is a very expensive and time-consuming work. Therefore, only very large companies like Yahoo! and Microsoft who offer search service have the capacity to collect such datasets [4, 25]; (ii) Gathering data from private documents like emails and having experts annotate them is unethical and would constitute a serious breach of privacy [34]; (iii) Annotated data often disagrees with the preferences of many users [29].

Therefore, practitioners in the field of LTR often use user interaction data, specifically user click data, as a common alternative to annotated datasets for training their models. In these methods, we do not know the true relevance labels, but we can infer relevance indicators from click data. User interaction data solves some of the problems associated with annotated datasets.: (i) Interaction data is free and can be collected by any service provider [26]; (ii) Interaction data can be collected without showing experts sensitive items for annotation [34]; (iii) Interaction data reflect the preferences of each user. However, interaction data also has specific disadvantages: (i) It requires the use of cookies to track the user's browsing activity (such as clicking on specific buttons, logging in, or recording history), which may violate user privacy; (ii) Click data is a very noisy signal because human behavior is unpredictable, users may click or not click for various reasons [5]; (iii) Relevance indicators from click data are also biased: (1) Position bias: users are more likely to examine, and click higher ranked items [6]. (2) Trust bias: users are more likely to click on on higher-ranked items that they do not actually like [1]. (3) Item-selection bias: not all the documents can be displayed in one ranking [21].

Researchers proposed Unbiased Learning to Rank (ULTR) [16] to learn unbiased ranking models from biased user interaction data, such as click data. There are two main families of methods for unbiased learning to rank: Counterfactual Learning to Rank (Counterfactual LTR) and Online Learning to Rank (Online LTR). Online LTR learns from real-time interaction with users [38, 20], while Counterfactual LTR learns from historical interactions [34, 12]. These two methods have been studied separately until Oosterhuis et al. [22] proposed a novel intervention-aware estimator that is designed for both online and counterfactual LTR. The authors demonstrate that the intervention-aware estimator outperforms existing estimators in terms of performance. Furthermore, the intervention-aware estimator's data efficiency increases when online intervention occurs. This means that with more interventions, less data is required to achieve optimal performance or, with the same amount of data, more interventions will lead to higher performance.

1.2 Research Questions

In this thesis, I explore the potential of making the intervention-aware estimator, a previously proposed approach for learning from user interactions, more efficient. While the intervention-aware estimator is able to reach performance comparable to state-of-the-art online and counterfactual LTR methods [22], it is not without limitations and challenges. One significant issue is the heavy computational load associated with the method. This limits the number of interventions that can be practically applied, thereby reducing data efficiency and algorithm performance. As a result, this approach may not be suitable for online applications that need to respond quickly to user preferences. The lengthy update process, which takes hours to complete, may make this approach impractical for real-time online applications.

To address these limitations and enhance the practicability of intervention-aware estimators, this thesis focuses on optimizing the method to reduce computational load and overall runtime without compromising performance. By monitoring and analyzing the runtime of various steps, including bias estimation and model training, I am able to identify bottlenecks and areas for improvement. This study aims to explore various methods for reducing the time required for bias estimation. Additionally, I delve into potential strategies to accelerate the model training process. To see how reduced runtime translates into performance gains in realistic settings, I introduce a novel experimental setup that simulates the effect of interventions on real-time online LTR. By addressing these research questions, I aim to contribute to the development of fast and accurate intervention-aware counterfactual estimators for online learning to rank.

There are several opportunities to save time in this algorithm. One approach to achieve this is through the monitoring and analysis of the runtime of the entire process, including the individual components such as bias estimation and model training. After identifying bottlenecks and areas for improvement, targeted solutions can be implemented to reduce overall runtime. Accordingly, the first research question of this thesis is:

RQ1 How much time does each step of the intervention-aware approach use while learning from (simulated) user interactions?

The analysis shows that bias estimation and model training are the most time-consuming components, the rest of the thesis will focus on how to improve the efficiency of these parts, aiming to significantly reduce the overall runtime of the algorithm.

RQ2 Among the five bias estimation methods—Frequency Estimation, Plackett-Luce Estimation, Direct Estimation Using Displayed Rankings, Utilizing Displayed Rankings with Additional Sampled Rankings, and Computing Expectations Based on Query Frequency— which method has the shortest runtime for bias estimation while maintaining the performance?

I delve into a series of strategies to improve the efficiency of the bias estimation process. This exploration includes different numbers of sampled rankings, different estimation methods, different data for estimation and innovative ways of computing expectations. The goal of these methods is to significantly reduce the time required for bias estimation while maintaining or enhancing result accuracy.

After determining the most effective algorithm for bias estimation, we use this as a foundation to explore various approaches for reducing the time required for model training.

RQ3 Among approaches like setting different maximum epoch values, iterative training with and without regularization techniques such as weight decay, which techniques yield the most efficient model training process while upholding or enhancing result accuracy?

These approaches include, but are not limited to, setting different maximum epoch values, implementing different model initialization strategies, such as starting from scratch or using an iterative approach. By carefully evaluating the impact of these approaches on runtime and model performance, I aim to identify the most effective methods for accelerating the model training process while maintaining or improving the accuracy of the results.

Through the optimization of the bias estimation and model training processes, I identify the most effective methods for saving the time for these steps. However, in my experimental setup, I do not consider the time required to generate user interaction data. In a real-world interactive system, the user's interaction with the system and the update of the system's logging policy are independent processes. To see how reduced runtimes can translate into performance gains in realistic settings, I introduce a novel experimental setting that simulates the effect of interventions on real-time online LTR.

RQ4 Does the reduced running time of the intervention-aware approach result in a meaningful difference in responsiveness?

By doing so, I aim to get insight into how efficiency interventions could translate into a faster and better performance thus a more responsive user experience with the system due to the more frequent updating of the system.

2 BACKGROUND

2.1 Learning to Rank

Learning to rank is an application of machine learning to build effective ranking models. As the name implies, it optimizes a ranking model by applying machine learning methods. The model should rank the documents in order of relevance, preference, or importance [13]. A ranking model is simply a ranking function that scores candidate documents given features about the query-document pairs and then sorts them in descending order based on the score. More formally, given a collection of documents D and a query q, a ranking system orders the documents $d \in D$ according to a scoring function f_θ. The goal of learning to rank is to find the parameters θ for f_θ that leads to the optimal ranking.

The existing learning to rank algorithms can be divided into three approaches: pointwise, pairwise, and listwise approaches. The pointwise approaches directly use regression or classification based algorithms to predict the exact value of relevance for each document. The pairwise approaches consider the relative order between two documents, which can be seen as a binary classification task. The listwise approaches try to optimize the ranking metrics of the whole ranking, such as NDCG and so on. Experiments comparing different learning to rank algorithms on several benchmark datasets show that listwise approaches generally outperforms pairwise and pointwise approaches [32].

2.2 Supervised Learning to Rank

Generally, LTR methods assume that each document d has a relevance label with respect to a query q [13], which can be modeled as the probability that a user thinks item d is relevant to query q: $P(R = 1 \mid q, d)$. In the LTR field, we think of the ranking system as a scoring function that assigns a score to each candidate document based on a given query and the ranking features of the document. Subsequently, these documents are ordered to form a ranking y. Hence, the primary goal is to learn an optimal scoring function $m(d)$ so that the most relevant documents are placed at the top of the ranking.

Let π be a ranking policy used to generate rankings for users, which can be understood as a probabilistic distribution over rankings given a specific query. In this context, $\pi(y \mid q)$ is the probability that the ranking policy π will present ranking y to the user for the given query q. Ideally it is important to consider all possible rankings that a policy can generate. The reward associated with a single ranking is denoted by $\mathcal{R}(y)$. Now, let $\mathcal{R}(q)$ represent the metric reward for query q under a ranking policy π, which is computed as the expected reward for all possible rankings:

$$\mathcal{R}(q) = \sum_{y \in \pi} \pi(y \mid q)\mathcal{R}(y). \tag{1}$$

The problem of learning to rank is often defined as an optimization problem for ranking metrics. $\mathcal{R}(y)$ can represent different relevance ranking metrics by choosing different θ_d, for example, Discounted Cumulative Gain (DCG):

$$\mathcal{R}(y) = \sum_{d \in y} \theta_d P(R = 1 \mid q, d), \quad \theta_d = \frac{1}{log_2\big(rank(d \mid y) + 1\big)}. \tag{2}$$

Then the overall metric reward $\mathcal{R}$ of this ranking policy π usually has the following form, which can be understood as the expected reward of a ranking system over the distribution of queries searched by users, in which $P(q)$ is the probability that a user searches for query q:

$$\mathcal{R} = \mathbb{E}_q[\mathcal{R}(q)] = \sum_q P(q)\mathcal{R}(q). \tag{3}$$

Thus, supervised LTR will try to optimize π to maximize $\mathcal{R}$, given the relevance label $P(R = 1 \mid q, d)$ [13, 35]. However, in practice, the relevance label is difficult to obtain directly.

2.3 Plackett-Luce Ranking Model and PL-Rank Algorithm

As mentioned in Section 2.2 about the distribution over rankings, the Plackett-Luce (PL) ranking model [23, 14] is often used to model this distribution [3, 8, 18, 17]. In the PL ranking model, $m(d)$ represents the score of a certain item d, then the probability of d is chosen to be the kth item in ranking y from the set of items D is:

$$\pi(d \mid y_{1:k-1}, D) = \frac{e^{m(d)}\mathbb{1}[d \notin y_{1:k-1}]}{\sum_{d' \in D \setminus y_{1:k-1}} e^{m(d')}}. \tag{4}$$

So the probability of the overall ranking is the product of the placement probabilities of each individual item:

$$\pi(y) = \prod_{k=1}^{K} \pi(y_k \mid y_{1:k-1}, D). \tag{5}$$

Unlike deterministic ranking models, the PL ranking model is possible for LTR methods to optimize ranking metrics directly, since the PL ranking model is fully differentiable. Typically, for a single query, the metric reward $\mathcal{R}(q)$ is calculated as I show in Equation 1, where ρ_{y_k} represents the relevance label the relevance label of the k-th item in ranking y, and the relevance is weighted by θ_k. $\mathcal{R}(q)$ can represent different kinds of relevance ranking metrics by choosing different θ_k, for example, if $\theta_k = \frac{1}{\log_2(k+1)}$, then $\mathcal{R}(q)$ will be the expectation of $DCG@K$ on the ranking policy π:

$$\mathcal{R}(q) = \sum_{y \in \pi} \pi(y \mid q)\mathcal{R}(y) = \sum_{y \in \pi} \pi(y \mid q) \underbrace{\sum_{k=1}^{K} \theta_k \rho_{y_k}}_{DCG@K}. \tag{6}$$

To maximize $R(q)$, the gradient of $\mathcal{R}(q)$ w.r.t. the general scoring function m is:

$$\frac{\delta}{\delta m}\mathcal{R}(q) = \sum_{d \in D} \left[\frac{\delta}{\delta m} m(d)\right] \cdot \lambda_d, \tag{7}$$

in which,

$$\lambda_d = \mathbb{E}_y \left[\left(\sum_{k=\mathrm{rank}(d,y)}^{K} \theta_k \rho_{y_k} \right) - \sum_{k=1}^{rank(d,y)} \pi(d \mid y_{1:k-1}) \left(\sum_{x=k}^{K} \theta_x \rho_{y_x} \right) \right]. \tag{8}$$

This is the PL-Rank-1 algorithm [17]. In practice, it is infeasible to compute this gradient exactly since λ_d is the expectation over all possible rankings. Luckily, we can first estimate λ_d using N sampled rankings, each ranking is of length K, and then estimate the gradient for all D items, so the computational complexity is $O(D \cdot N \cdot K)$. In order to improve the sample efficiency, Oosterhuis also proposed the PL-Rank-2 algorithm [17] with same computational complexity by rewriting the λ_d:

$$\lambda_d = \mathbb{E}_y \left[\left(\sum_{k=rank(d,y)+1}^{K} \theta_k \rho_{y_k} \right) + \sum_{k=1}^{rank(d,y)} \pi(d \mid y_{1:k-1}) \left(\theta_k \rho_d - \sum_{x=k}^{K} \theta_x \rho_{y_x} \right) \right]. \tag{9}$$

In Oosterhuis's subsequent work [18], he proposed the PL-Rank-3 algorithm by using the following property:

$$\sum_{k=1}^{\text{rank}(d,y)} \pi(d \mid y_{1:k-1}) = e^{f(d)} \left(\sum_{k=1}^{\text{rank}(d,y)} \frac{1}{\sum_{d' \in D \setminus y_{1:k-1}} e^{f(d')}} \right), \tag{10}$$

and three new vectors: $PR_{y,d}$, $DR_{y,d}$ and $RI_{y,d}$:

$$\begin{aligned} PR_{y,i} &= \sum_{k=i}^{\min(i,K)} \theta_k \rho_{y_k}, \quad PR_{y,d} = PR_{y,rank(d,y)+1}, \\ RI_{y,i} &= \sum_{k=1}^{\min(i,K)} \frac{PR_{y,k}}{\sum_{d' \in D \setminus y_{1:k-1}} e^{f(d')}}, \quad RI_{y,d} = RI_{y,\text{rank}(d,y)}, \\ DR_{y,i} &= \sum_{k=1}^{\min(i,K)} \frac{\theta_k}{\sum_{d' \in D \setminus y_{1:k-1}} e^{f(d')}}, \; DR_{y,d} = DR_{y,\text{rank}(d,y)}, \end{aligned} \tag{11}$$

which enable:

$$\lambda_d = \mathbb{E}_y \left[PR_{y,d} + e^{f(d)} \left(\rho_d DR_{y,d} - RI_{y,d} \right) \right]. \tag{12}$$

The novel PL-Rank-3 algorithm computes the same approximation with $O(D \cdot (N + K))$ given N sampled rankings.

2.4 Unbiased Learning to Rank

Supervised LTR comes with several limitations due to its reliance on supervised learning and expert annotations, for example, finding the true relevance label $P(R = 1 \mid q, d)$ is quite infeasible in many practical cases [34].

In response to this challenge, practitioners turn to user interaction data, particularly user click data, as a common alternative for model training. Despite not knowing the true relevance labels, these methods infer relevance indicators from the user interactions. User interaction data offers various advantages. Firstly, it is freely available and can be collected by any service provider without the need for expert intervention [26]. Secondly, gathering interaction data does not involve exposing sensitive content to experts, thus preserving user privacy [34]. Lastly, this data more closely matches the preferences of individual users [11]. However, click data is inherently biased. Several biases, such as position bias (users are more likely to examine, and click higher ranked items [6]), trust bias (users are more likely to click on on higher-ranked items that they do not actually like [1]), and item-selection bias (not all the documents can be displayed in one ranking [21]), prevent us from simply using click data as relevance labels.

To learn unbiased ranking models from such biased user interaction (click) data, researchers proposed Unbiased Learning to Rank (ULTR) [16]. There are two main families of methods for unbiased learning to rank: Counterfactual Learning to Rank (Counterfactual LTR) and Online Learning to Rank (Online LTR). Online LTR learns from real-time interaction with users [38, 20], while Counterfactual LTR learns from historical interactions [34, 12].

2.5 Click Models for Web Search

Click models are a class of statistical models used to understand user behavior in web search with a ranked list of items. It is very important to developing click models.

Firstly, by modeling user behavior with click data, we can gain a better understanding of users, ultimately leading to an improved search experience. Moreover, such a model can be used to predict document relevance for items not directly observed, as we do in Counterfactual LTR. Furthermore, experiments with real users may not always be feasible in many cases. In such situations, accurate user simulations based on well-defined click models become very important. In summary, click models make user clicks in web search easier to understand user behavior, build evaluation metrics, approximate document relevance, and perform simulation experiments by using probabilistic graphical models [5].

There are several types of click models, each with its own assumptions. Some common click models include random click model (RCM) [5], rank-based click model (RCTR) [10], document-based click model (DCTR) [6] and position-based click model (PBM) [5, 6]. The random click model (RCM) [5] is a simple and widely used click model, which assumes that users click on documents randomly, without considering the relevance of the documents. In other words, regardless of the quality of the ranking, each document is assigned an equal probability of being clicked. This sharply contrasts with the rank-based click model (RCTR) [10] and document-based click model (DCTR) [6], which respectively assume that the click probability of a document depends only on its rank or its relevance. While the Position-Based Model (PBM) follows the examination hypothesis, which states that users will click on a document if and only if they examine it and are attracted to it (and therefore click after having examined it) [5, 6]:

$$P(C=1) = P(E=1) \cdot P(A=1) = P(E=1) \cdot P(C=1 \mid E=1), \tag{13}$$

where $P(E=1)$ is the examination probability and $P(C=1 \mid E=1)$ is the conditional probability of clicking on the document given that it is examined.

It should be emphasized that attractiveness here refers to the true or perceived relevance of the document, so the probability of a document being clicked can be factored as:

$$\begin{aligned} P(C=1) &= P(E=1) \cdot P(C=1 \mid E=1) \\ &= \begin{cases} P(E=1 \mid k) \cdot P(R=1 \mid q,d), & R \text{ is true relevance} \\ P(E=1 \mid k) \cdot P(\tilde{R}=1 \mid E=1,q,d). & \tilde{R} \text{ is perceived relevance} \end{cases} \end{aligned} \tag{14}$$

Joachims et al. [10] show that the examination probability of the document often depends heavily on its rank or position k, while the true relevance R (perceived relevance $\tilde{R}$) depends on both the query q and document d by introducing query-document pairs. A probabilistic graphical model representation of this model is shown in Figure 1.

2.6 Counterfactual Learning to Rank

Counterfactual LTR algorithms leverage historical user interactions. Instead of considering clicks directly as biased relevance indicators, these algorithms re-weight clicks to debias potential biases present in the interaction data [9]. Through this approach, we learn a ranking model that aims to mitigate the impact of biases and provide more accurate relevance indicators.

2.6.1 Position Bias

Position bias, in the context of information retrieval, means users are more likely to click on or interact with documents that are presented at the top of the ranking list, regardless of their actual relevance or quality. To correct for position bias to ensure fair and unbiased rankings, Wang et al. [34] and Joachims et al. [12] proposed the Inverse

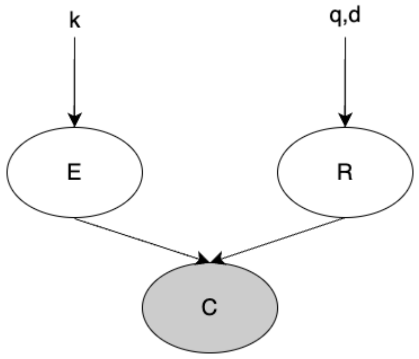

Figure 1: Graphical representation of the position-based model. E is the random variable for examination, R is the random variable for true relevance, k means position, rank, d is document identifier, q is query identifier.

Propensity Scoring based estimator, which re-weight clicks according to the examination probabilities.

As I mentioned before, in a basic Position-Based Model (PBM), a click depends jointly on the examination E and relevance R:

$$\begin{aligned} P(C = 1 \mid q, y, d) &= P(E = 1 \mid q, y, d) \cdot P(C = 1 \mid E = 1, q, y, d) \\ &= P(E = 1 \mid k) \cdot P(R = 1 \mid q, d). \end{aligned} \tag{15}$$

The main idea is that if the examination probability $P(E = 1 \mid k)$ is positive, the position bias for each click can be corrected. Intuitively, this condition exists because documents that cannot be examined can never receive clicks [21]. Thus, the relevance indicator of IPS estimator is:

$$P(R = 1 \mid q, d) = \frac{P(C = 1 \mid q, y, d)}{P(E = 1 \mid k)}. \tag{16}$$

Additionally, Joachims et al. [12] proved that it is still unbiased under click noise. In fact, all recent counterfactual LTR methods are based on this IPS estimator.

2.6.2 Item-Selection Bias

The IPS estimator described in Section 2.6.1 is only unbiased if there is no item-selection bias. That is, all relevant documents need to have a non-zero examination probability in every ranking, which means that all relevant documents must be displayed to the users in every ranking. However, in real search engines, the most common situation is that the system is set to only display the top-K documents, and users cannot examine documents other than top-K.

To correct for this bias, we have to consider all possible rankings under the logging policy π. Because this way, even if all relevant documents cannot be displayed in one top-K ranking at the same time, they may be examined by users in different top-K

rankings. This leads to a policy-aware estimator [21]:

$$\begin{aligned}
P(C=1 \mid q, \pi, d) &= \sum_y \pi(y \mid q) \cdot P(C=1 \mid q, y, d) \\
&= \sum_y \pi(y \mid q) \cdot P(E=1 \mid q, y, d) \cdot P(C=1 \mid E=1, q, y, d) \\
&= \sum_y \pi(y \mid q) \cdot P(E=1 \mid y, d) \cdot P(R=1 \mid q, d) \\
&= P(R=1 \mid q, d) \cdot \sum_y \pi(y \mid q) \cdot P(E=1 \mid y, d).
\end{aligned} \tag{17}$$

Thus, the relevance indicator of policy-aware estimator is:

$$P(R=1 \mid q, d) = \frac{P(C=1 \mid q, \pi, d)}{\sum_y \pi(y \mid q) \cdot P(E=1 \mid y, d)}. \tag{18}$$

2.6.3 Trust Bias

There also may be situations where users trust the ranking system and are therefore more likely to click on documents they perceive as relevant, even if they are not actually relevant. This is what we call a trust bias [10]. Vardasbi et al. [33] prove that IPS estimator cannot correct for trust bias. As an alternative, they introduce an affine estimator based on the idea of perceived relevance $\tilde{R}$:

$$\begin{aligned}
P(C=1 \mid q, y, d) =& P(E=1 \mid q, y, d) \cdot P(C=1 \mid E=1, q, y, d) \\
=& P(E=1 \mid k) \cdot P(\tilde{R}=1 \mid E=1, q, y, d) \\
=& P(E=1 \mid k) \cdot \sum_{R \in \{0,1\}} P(\tilde{R}=1, R \mid E=1, q, y, d) \\
=& P(E=1 \mid k) \cdot \sum_{R \in \{0,1\}} P(\tilde{R}=1 \mid E=1, R, k) \cdot P(R \mid q, d).
\end{aligned} \tag{19}$$

To simplify the notation, I follow Vardasbi et al. [33] and adopt:

$$\begin{aligned}
\alpha_k &= P(E=1 \mid k) \cdot \Big[P(\tilde{R}=1 \mid E=1, R=1, k) - P(\tilde{R}=1 \mid E=1, R=0, k)\Big], \\
\beta_k &= P(E=1 \mid k) \cdot P(\tilde{R}=1 \mid E=1, R=0, k),
\end{aligned} \tag{20}$$

the Eq. 19 can be simplified to

$$P(C=1 \mid q, y, d) = \alpha_k \cdot P(R=1 \mid q, d) + \beta_k. \tag{21}$$

This affine estimator penalizes the document displayed at rank k by β_k while also reweighting by $\frac{1}{\alpha_k}$:

$$P(R=1 \mid q, d) = \frac{P(C=1 \mid q, y, d) - \beta_k}{\alpha_k}. \tag{22}$$

2.6.4 Intervention-Oblivious Estimator

However, while the affine estimator correct for position bias and trust bias, it cannot correct for item-selection bias, the policy-aware estimator correct for position bias and item-selection bias, it cannot correct for trust bias. Oosterhuis et al. [22] introduced an intervention-aware estimator, which is the first to effectively correct for all three biases simultaneously.

Before introducing intervention-aware estimator, I want to introduce intervention-oblivious estimator first. The intervention-oblivious estimator [22] combines a policy-aware estimator with an affine estimator, meaning it also corrects for position bias, trust bias, and item-selection bias simultaneously. In the estimator, we need to consider both π and $\tilde{R}$:

$$\begin{aligned}
P(C = 1 \mid q, \pi, d) &= \sum_y \pi(y \mid q) \cdot P(C = 1 \mid q, y, d) \\
&= \sum_y \pi(y \mid q) \cdot P(E = 1 \mid q, y, d) \cdot P(C = 1 \mid E = 1, q, y, d) \\
&= \sum_y \pi(y \mid q) \cdot P(E = 1 \mid y, d) \cdot P(\tilde{R} = 1 \mid E = 1, q, y, d) \\
&= \sum_y \pi(y \mid q) \cdot P(E = 1 \mid y, d) \cdot \sum_{R \in \{0,1\}} P(\tilde{R} = 1, R \mid E = 1, q, y, d).
\end{aligned} \tag{23}$$

Simplify it in the same way:

$$\begin{aligned}
&\alpha_{y,d} = P(E = 1 \mid y, d) \cdot \left[P(\tilde{R} = 1 \mid E = 1, R = 1, y, d) - P(\tilde{R} = 1 \mid E = 1, R = 0, y, d) \right], \\
&\beta_{y,d} = P(E = 1 \mid y, d) \cdot P(\tilde{R} = 1 \mid E = 1, R = 0, y, d).
\end{aligned} \tag{24}$$

Finally, we get the intervention-oblivious estimator:

$$\begin{aligned}
P(R = 1 \mid q, d) &= \frac{P(C = 1 \mid q, \pi, d) - \sum_y \pi(y \mid q)\beta_{y,d}}{\sum_y \pi(y \mid q)\alpha_{y,d}} \\
&= \frac{P(C = 1 \mid q, \pi, d) - \mathbb{E}_y[\beta_{y,d} \mid \pi, q]}{\mathbb{E}_y[\alpha_{y,d} \mid \pi, q]}.
\end{aligned} \tag{25}$$

2.6.5 Intervention-Aware Estimator

In the intervention-oblivious estimator, we only consider one single policy π, which can be understood as the method is designed for the scenario where the logging policy is static, that is, the logging policy is the same at any time steps. In a real search engine, this is obviously impossible. What we need to consider is that the logging policy may be updated at any time when collecting data.

Although Oosterhuis et al. [22] have proved that even in this case the intervention-oblivious estimator is still unbiased, the authors also mentioned that we still cannot ignore deploying different logging policies at different time steps. So they proposed the intervention-aware estimator in the same paper [22], which takes into account all logging policies: $\Pi_T = \{\pi_1, \pi_2, ..., \pi_T\}$ employed to collect the data. Notably, this estimator also successfully mitigates position bias, item-selection bias, and trust bias:

$$\begin{aligned}
P(R = 1 \mid q, d) &= \frac{P(C = 1 \mid q, \Pi_T, d) - \frac{1}{T}\sum_{t=1}^{T} \sum_y \pi_t(y \mid q)\beta_{y,d}}{\frac{1}{T}\sum_{t=1}^{T} \sum_y \pi_t(y \mid q)\alpha_{y,d}} \\
&= \frac{P(C = 1 \mid q, \Pi_T, d) - \mathbb{E}_{t,y}[\beta_{y,d} \mid \Pi_T, q]}{\mathbb{E}_{t,y}[\alpha_{y,d} \mid \Pi_T, q]}.
\end{aligned} \tag{26}$$

A notable difference from intervention-oblivious estimator is that in intervention-aware estimator the expectation is over all logging policies Π_T, rather than a single logging policy π. Although this difference may seem subtle, experimental results show that there is a huge performance difference. It is for this reason that my thesis focuses on making intervention-aware estimators better.

For clarity, I will present its algorithmic details in Algorithm 1. Its input requires a starting policy (π_0), a choice of ranking metric θ (Equation 6), the α and β parameters (Equation 24), a set of intervention timesteps Φ, and the final timestep T.

Algorithm 1 Original Intervention-Aware Estimator[22]

```
1: Input: Starting policy: $\pi_0$; Metric weight function: $\theta$;
          Inferred bias parameters: $\alpha$ and $\beta$;
          Interventions steps: $\Phi$; End-time: $T$.
2: $\mathcal{D} \leftarrow \{\}$                                   // initialize data container
3: $\pi \leftarrow \pi_0$                                    // initialize logging policy
4: for $i \in \Phi$ do
5:     $\mathcal{D} \leftarrow \mathcal{D} \cup \text{gather}(\pi, i- \mid \mathcal{D} \mid)$          // observe i− | $\mathcal{D}$ | timesteps
6:     $\pi \leftarrow \text{optimize}(\mathcal{D}, \alpha, \beta, \pi_0, \theta)$        // optimize based on available data
7: $\mathcal{D} \leftarrow \mathcal{D} \cup \text{gather}(\pi, T- \mid \mathcal{D} \mid)$                // expand data to T
8: $\pi \leftarrow \text{optimize}(\mathcal{D}, \alpha, \beta, \pi_0)$              // optimize based on final data
9: return $\pi$
```

The algorithm starts by initializing an empty set to store the gathered interaction data (Line 2) and initializes the logging policy with the provided starting policy π_0 (Line 3). Then for each timestep i in Φ, from last intervention timestep $|\mathcal{D}|$ to current intervention timestep i, the dataset is expanded using the current logging policy π to display rankings to users, and the resulting interactions are added to $\mathcal{D}$ (Line 5). Then the starting policy π_0 is optimized using the available data in $\mathcal{D}$, which will be the new logging policy π (Line 6). Each iteration results in an intervention that generates a new policy that replaces the old logging policy, changing the way future data is logged. After the Φ iterations are done, more data is collected so that $|\mathcal{D}| = T$ (Line 7)and the optimization is performed one last time (Line 8). The resulting policy is the final result of this process.

3 METHODS

In this chapter, I will discuss the methodology of this thesis. The thesis builds upon the intervention-aware estimator as introduced in Section 2.6.5. This work forms the basis for the subsequent optimization work and analysis in this thesis.

In my method, I will use the same inputs and follow most of the steps and settings from Algorithm 1. What I will focus on modifying and improving is the optimization process (Line 6), which is to use the collected data to optimize the starting policy and get a new logging policy. The algorithm will go through this optimization in each iteration (intervention), if it can be made more effective, it will greatly improve the efficiency of the entire algorithm, allowing more interventions to be run.

The optimization process (Line 6) is divided into two main steps in Algorithm 2. The first step is bias estimation, which is used to derive the relevance indicator from the click data: $P(R = 1 \mid q, d) = \rho_d$. The second step is model training, using the relevance indicator ρ_d derived from the previous step as the relevance label, and then follow supervised LTR (PL-Rank-3) to train the ranking model. I will focus on improving these two steps.

Algorithm 2 Optimization Process

```
1: Input: Starting policy: $\pi_0$; Metric weight function: $\theta$;
          Inferred bias parameters: $\alpha$ and $\beta$; Click data: $\mathcal{D}$;
          Logging policys: $\Pi_T = \{\pi_1, \pi_2, ..., \pi_T\}$.
2: $\rho_d \leftarrow$ estimate$(\mathcal{D}, \alpha, \beta, \Pi_T)$          // estimate relevance indicator
3: $\pi \leftarrow$ PL-Rank-3$(\alpha, \beta, \rho_d, \pi_0, \theta)$          // train a new logging policy
4: return $\pi$
```

3.1 Bias Estimation Efficiency

In the original intervention-aware estimator, in order to derive the unbiased relevance indicator, we need to first calculate the expected α and β over all logging policies $\Pi_T = \{\pi_1, \pi_2, ..., \pi_T\}$:

$$\begin{aligned} P(R = 1 \mid q, d) &= \frac{P(C = 1 \mid q, \Pi_T, d) - \mathbb{E}_{t,y}[\beta_{y,d} \mid \Pi_T, q]}{\mathbb{E}_{t,y}[\alpha_{y,d} \mid \Pi_T, q]} \\ &= \frac{P(C = 1 \mid q, \Pi_T, d) - \frac{1}{T}\sum_{t=1}^{T}\sum_{y} \pi_t(y \mid q)\beta_{y,d}}{\frac{1}{T}\sum_{t=1}^{T}\sum_{y} \pi_t(y \mid q)\alpha_{y,d}}. \end{aligned} \tag{27}$$

However, for each logging policy π_t, it is computationally infeasible to exactly compute the probability $\pi_t(y \mid q)$ of all possible rankings, so we need to transform the original equation into:

$$\begin{aligned} P(R = 1 \mid q, d) &= \frac{P(C = 1 \mid q, \Pi_T, d) - \mathbb{E}_{t,y}[\beta_{y,d} \mid \Pi_T, q]}{\mathbb{E}_{t,y}[\alpha_{y,d} \mid \Pi_T, q]} \\ &= \frac{P(C = 1 \mid q, \Pi_T, d) - \frac{1}{T}\sum_{t=1}^{T}\sum_{k \in \{1,2,...,K\}} \pi_t(k \mid q, d)\beta_k}{\frac{1}{T}\sum_{t=1}^{T}\sum_{k \in \{1,2,...,K\}} \pi_t(k \mid q, d)\alpha_k}. \end{aligned} \tag{28}$$

For a single ranking y for query q, let k be the rank at which document d is displayed in ranking y, so $\alpha_{y,d} = \alpha_k$, $\beta_{y,d} = \beta_k$. This allows us to convert the probability of the ranking y under current logging policy $\pi_t(y \mid q)$ into the probability that document d is ranked as the k-th document under current logging policy $\pi_t(k \mid q, d)$, which is something that we can estimate with a practical sampling strategy. In my thesis, following Equation 28, I explore various different methods to improve the efficiency and accuracy of the estimation.

3.1.1 Frequency Estimation (Original Intervention-Aware Estimator)

To estimate $\pi_t(k \mid q, d)$, Oosterhuis et al. [22] propose to sample N additional rankings for every query under the current logging policy, and calculate the probability that each preselected document of the query is ranked in different positions through frequency estimation. Every time the logging policy is updated (intervention), N additional rankings are sampled again for estimation, all probabilities are updated accordingly. It is important to note that, often in real-world scenarios, users will only be shown the top-K results, which means, it is impossible to include all preselected documents in a single ranking. This is illustrated in the following example in Table 1.

Ranking	1st (y_1)	2nd (y_2)	3rd (y_3)
$y^{(1)}$	d_1	d_2	d_3
$y^{(2)}$	d_2	d_1	d_3
$y^{(3)}$	d_1	d_2	d_3

Table 1: Sampled Rankings

Assuming a query q has four preselected documents $\{d_1, d_2, d_3, d_4\}$, three additional rankings are sampled for estimation. If only relying on frequency estimation, the probability of d_4 being ranked in each position will be 0, which would make the denominator in Equation 28 equal to 0. The solution is to use frequency estimation for all positions except the last one, while the probability of the last position must be calculated according to the Plackett-Luce ranking model. In the Plackett-Luce (PL) ranking model, $m(d)$ represents the score of a certain document d, then the probability of d is chosen as the k-th item in ranking y from the set of documents D is:

$$\pi(d \mid y_{1:k-1}, D) = \frac{e^{m(d)} \mathbb{1}[d \notin y_{1:k-1}]}{\sum_{d' \in D \setminus y_{1:k-1}} e^{m(d')}}. \tag{29}$$

Specifically, when $k < K$, the estimation is the observed frequency: $\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[d = y_k^{(i)}]$; when $k = K$, the estimation is the expectation of Equation 29 over the observed rankings: $\frac{1}{N} \sum_{i=1}^{N} \pi(d \mid y_{1:k-1}^{(i)}, D)$. Finally, the probabilities of ranking each document at different positions are listed in Table 2.

Doc	$\pi(k = 1 \mid q, d)$	$\pi(k = 2 \mid q, d)$	$\pi(k = 3 \mid q, d)$
d_1	$\frac{2}{3}$	$\frac{1}{3}$	0
d_2	$\frac{1}{3}$	$\frac{2}{3}$	0
d_3	0	0	$\frac{e^{m(d_3)}}{e^{m(d_3)}+e^{m(d_4)}}$
d_4	0	0	$\frac{e^{m(d_4)}}{e^{m(d_3)}+e^{m(d_4)}}$

Table 2: Frequency Estimation

It should be pointed out that using this method, many probabilities will be zero when $k < K$, which may lead to underestimation of certain propensities, or at least inaccurate estimates.

3.1.2 Plackett-Luce Estimation

To explore Research Question 2 in section 1.2, I first follow the original intervention-aware estimator to generate N additional rankings for estimation. However, instead of using frequency estimation for positions $k < K$ like in the original method, I employ the Plackett-Luce ranking model for each position k to exactly calculate the probability of selecting document d as the k-th document in each sampled ranking y and then average them. That is, for each k, the estimation is the expectation of Equation 29 over the observed rankings: $\frac{1}{N}\sum_{i=1}^{N}\pi(d \mid y^{(i)}_{1:k-1}, D)$.

I use the same sampled rankings in Table 1, according to the Plackett-Luce ranking model, the probability of each document being ranked in different positions is shown in Table 3.

Doc	$\pi(k=1 \mid q,d)$	$\pi(k=2 \mid q,d)$
d_1	$\frac{e^{m(d_1)}}{e^{m(d_1)}+e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}$	$\frac{1}{3}\cdot\frac{e^{m(d_1)}}{e^{m(d_1)}+e^{m(d_3)}+e^{m(d_4)}}$
d_2	$\frac{e^{m(d_2)}}{e^{m(d_1)}+e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}$	$\frac{2}{3}\cdot\frac{e^{m(d_2)}}{e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}$
d_3	$\frac{e^{m(d_3)}}{e^{m(d_1)}+e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}$	$\frac{2}{3}\cdot\frac{e^{m(d_3)}}{e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}+\frac{1}{3}\cdot\frac{e^{m(d_3)}}{e^{m(d_1)}+e^{m(d_3)}+e^{m(d_4)}}$
d_4	$\frac{e^{m(d_4)}}{e^{m(d_1)}+e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}$	$\frac{2}{3}\cdot\frac{e^{m(d_4)}}{e^{m(d_2)}+e^{m(d_3)}+e^{m(d_4)}}+\frac{1}{3}\cdot\frac{e^{m(d_4)}}{e^{m(d_1)}+e^{m(d_3)}+e^{m(d_4)}}$

Doc	$\pi(k=3 \mid q,d)$
d_1	0
d_2	0
d_3	$\frac{e^{m(d_3)}}{e^{m(d_3)}+e^{m(d_4)}}$
d_4	$\frac{e^{m(d_4)}}{e^{m(d_3)}+e^{m(d_4)}}$

Table 3: Plackett-Luce Estimation

In summary, the two methods mentioned above require a lot of additional rankings for estimation every time the logging policy is changed (intervention). This would obviously make the estimation inefficient. One solution is to sample as little as possible rankings, that is, to make N as small as possible. Another solution is to not sample the additional rankings at all.

3.1.3 Direct Estimation Using Displayed Rankings

At each time step, the previous methods uniformly sample a user-issued query and generate a ranking display to the user according to the logging policy, and user interactions are simulated on this ranking. However, it is wasted if the displayed rankings are only used to collect user interactions. In this way, I directly use the rankings displayed to the users during the interaction process to estimate $\pi(k \mid q, d)$ by means of frequency estimation, thus saving a lot of time.

3.1.4 Utilizing Displayed Rankings with Additional Rankings

In addition to simply using displayed rankings, it is also possible to extend the collected interaction data from one ranking to N rankings. This means that I only sample additional rankings for queries searched by users during the interaction process, not for every query. This method can be seen as a combination of generating new rankings and directly using existing rankings. In order to be able to compare with the original intervention-aware estimator, I still use frequency estimation in this approach.

3.1.5 Frequency Expectation Calculation

The three alternatives I mentioned above mainly consider using different methods to estimate $\pi(k \mid q, d)$, either changing the estimation algorithm or changing the data used for the estimation. The method I now propose will no longer consider estimating $\pi(k \mid q, d)$, but uses a completely new method to calculate the expectation of α and β, given $\pi(k \mid q, d)$ is known (which has been estimated by using the original intervention-aware estimator). In all previous methods, the expectation is computed over all logging policies $\Pi_T = \{\pi_1, \pi_2, ..., \pi_T\}$. However, in this approach, the expectation is computed over the distribution of user-issued queries. For example, when the logging policy is π_1, the user searches for q twice, and when the logging policy is π_2, the user searches for q once, then the expectations for α and β in Equation 28 are as follows:

$$\begin{aligned}\mathbb{E}_{t,y}[\beta_{y,d} \mid \Pi_T, q] &= \frac{2}{3} \sum_{k \in \{1,2,...,K\}} \pi_1(k \mid q, d)\beta_k + \frac{1}{3} \sum_{k \in \{1,2,...,K\}} \pi_2(k \mid q, d)\beta_k, \\ \mathbb{E}_{t,y}[\alpha_{y,d} \mid \Pi_T, q] &= \frac{2}{3} \sum_{k \in \{1,2,...,K\}} \pi_1(k \mid q, d)\alpha_k + \frac{1}{3} \sum_{k \in \{1,2,...,K\}} \pi_2(k \mid q, d)\alpha_k.\end{aligned} \tag{30}$$

The advantage of this method is that if a query is not searched by users, then there is no need to estimate the bias for this query, because it will be weighted by 0.

3.2 Model Training Efficiency

For Research Question 3, this thesis focuses on speeding up the model training process within an intervention-aware estimator. I explore different methods to save time in model training. One approach is to set a maximum epoch value. This can be a fixed value or a varying maximum epoch value. Another approach is using iterative training. In Line 3, the new logging policy π_t is obtained by training from scratch:

$$\pi_t \leftarrow \text{PL-Rank-3}(\alpha, \beta, \rho_d, \pi_0, \theta). \tag{31}$$

This means that we have to train from the starting policy π_0 every intervention, which can take a lot of time. Therefore, I propose trying iterative training, where the new logging policy π_t is trained from the last logging policy π_{t-1}:

$$\pi_t \leftarrow \text{PL-Rank-3}(\alpha, \beta, \rho_d, \pi_{t-1}, \theta). \tag{32}$$

I get this inspiration from online machine learning [36], which refers to the process of continuously updating model parameters as new data becomes available. This approach is particularly useful when dealing with streaming data or scenarios where the distribution of data may change over time, which is a good fit for the situation I am facing: at each intervention, newly arriving data will be collected by the newly deployed logging policy.

Iterative training builds on previous model, allowing the model to learn from new data without losing knowledge already learned. This can lead to faster convergence and fewer training epochs.

4 EXPERIMENTAL SETUP

In this thesis, I use the semi-synthetic experimental setup that is common in existing work on unbiased LTR [16, 21, 22, 20, 12, 9, 33]. In this setup, I simulate a web-search scenario by sampling queries and documents from a commercial search dataset, while user interactions and rankings are simulated using probabilistic click models. The advantage of this setup is that it allows us to investigate the effects of online interventions on a large scale while also being easy to reproduce by researchers without access to real search scenarios.

4.1 Dataset

I use the one of the largest publicly-available LTR industry datasets: Yahoo! Learning to Rank Challenge dataset [4]. This dataset consists of 29,921 queries with 709,877 query-document pairs. There are 699 features for query-document pairs, and five expert-judged relevance labels $\{0, 1, 2, 3, 4\}$. The dataset is divided into training set, test set and validation set. The training and validation sets are used to simulate training clicks and validation clicks, and the test set is used to evaluate the performance of the current (logging) policy.

4.2 Ranking Model

To start, I randomly select 20 queries from training set and use supervised LTR to optimize a ranking model as the start logging policy π_0 in Algorithm 1. The resulting start ranking model has much better performance than a randomly initialized model, yet still leaves room for improvement.

All ranking models are neural networks with two hidden layers, each containing 32 hidden units with sigmoid activations. Models are optimized using policy gradients estimated with PL-Rank-3 [18]. All policies apply a softmax to the document scores produced by the ranking models to match the Plackett-Luce ranking model. Clipping is only applied on the training clicks, denominators of any estimator are clipped by $10/\sqrt{|\mathcal{D}|}$ to reduce variance. Early stopping is applied based on counterfactual estimates of the loss using (unclipped) validation clicks.

4.3 Click Simulation

I simulate up to 10^7 clicks (aka time steps) per run. The number of training clicks and validation clicks is always chosen as the ratio between the number of training and validation queries in the dataset. The size of Φ (the intervention steps) is 50 per run, and the time steps in Φ are evenly spread on an exponential scale. At each time step, I simulate a user-issued query by uniformly sampling from the training or validation set. The preselected documents for this query are then ranked according to the current logging policy, and user interactions are simulated on the top-5 documents of the overall ranking using a probabilistic click model. I apply Equation 21 with $\alpha^{top-5} = [0.35, 0.53, 0.55, 0.54, 0.52]$ and $\beta^{top-5} = [0.65, 0.26, 0.15, 0.11, 0.08]$; the relevance probabilities $P(R = 1 \mid d, q) = 0.25 \cdot \text{relevance\_label}(d, q)$. The parameters of α and β are chosen based on previous work by Agarwal et al. [1], who inferred them from real-world user behavior. In doing so, our goal is to simulate an environment with realistic levels of position bias, item-selection bias, and trust bias.

4.4 Experiment Details

In order to select an appropriate number of interventions for all subsequent experiments, I first compare different number of interventions ($|\Phi| = \{10, 20, 50, 100, 200\}$) using the original intervention-aware estimator in Section 3.1.1. I then choose the most suitable 50 interventions as the basis for all subsequent experiments.

To address Research Question 1, I use the original intervention-aware estimator with $|\Phi| = 50$ interventions and $N = 1000$ sampled rankings as the baseline model. I measure the runtime of each part, including bias estimation and model training, as well as the total runtime.

For Research Question 2, I compare different numbers of sampled rankings ($N = \{10, 100, 1000\}$), different estimation methods (as described in Section 3.1.2), different data for estimation (as described in Sections 3.1.3 and 3.1.4) and different ways of computing expectations (as described in section 3.1.5).

For Research Question 3, I experiment with different maximum epoch values, both fixed and varying: $\text{max\_epochs} = \{6, \lfloor \log_{10}(|\mathcal{D}|) \rfloor - 1\}$. I also compare training from scratch with iterative training with and without regularization techniques such as weight decay.

In Research Question 4, in order to understand the potential performance gains of these accelerations in real-world settings, I introduce a novel experimental setup that simulates the impact of interventions on real-time learning curves. Specifically, during the process of bias estimation and model training, users are also generating data. This means that all click data generated during this period is used as training data for the next model training. I simulate three user interaction systems of different scales, that is, $\{1000, 1500, 3600\}$ logged queries are generated per second. I set up this simulation environment to observe the performance of the algorithm during the whole one-hour simulation.

Finally, it is important to note that each experimental result is the average of 20 independent runs. Figures plot the mean and shaded areas represent 90% confidence intervals. The performance metric used in all experiments is Normalized Discounted Cumulative Gain (NDCG).

5 EXPERIMENTAL RESULTS

First of all, in order to select an appropriate number of interventions for all subsequent experiments, I consider Figure 2 which displays the performance of the original intervention-aware estimator proposed by Oosterhuis et al. [22] with different number of interventions.

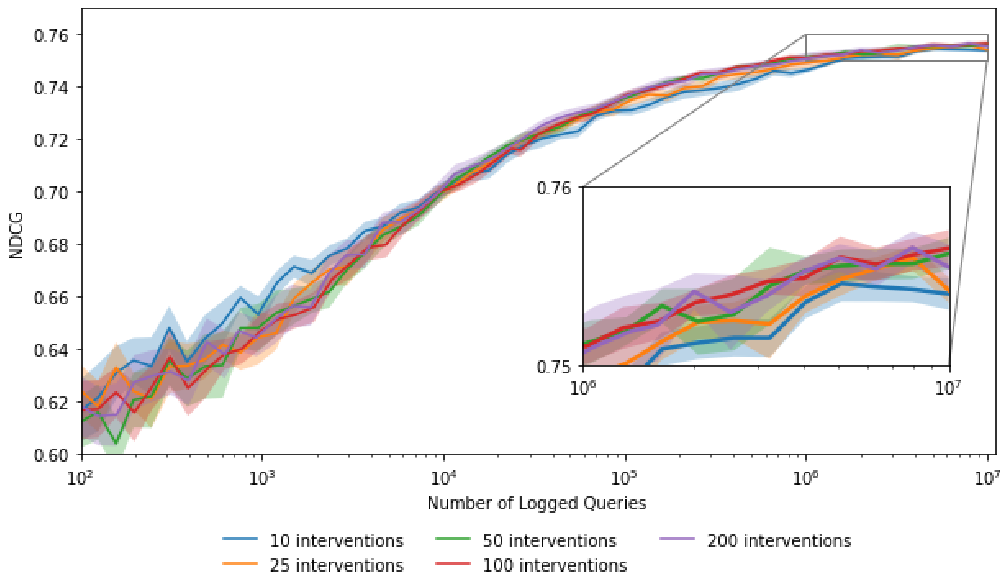

Figure 2: Comparison of different number of interventions. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

From this figure, we can see that as the number of logged queries (clicks) increases, the NDCG score also increases. This suggests that as more data is collected, the performance of the estimator improves. Additionally, we can see that higher numbers of interventions result in higher NDCG scores. In particular, the performance of 50, 100, and 200 interventions is better than that of 10 and 20 interventions most of the time. This suggests that using more interventions can ultimately improve the performance of the estimator. However, it is difficult to determine from the figure which of the three (50, 100, or 200 interventions) performs better. Therefore, considering that more interventions lead to longer runtime, I decide to use 50 interventions in all subsequent experiments, since it uses the least amount of time while achieving the optimal performance.

5.1 Runtime Analysis

To answer the Research Question 1: *how much time does each step of the intervention-aware approach use while learning from user interactions*, I consider Figure 3 which displays the runtime of each part of the optimization process in Algorithm 1, including bias estimation and model training, as well as the total runtime.

From this figure, we can observe that the total runtime is approximately 9500 seconds, of which about 3000 seconds are used for bias estimation and the remaining 6500 seconds are used for model training. Additionally, we can see that the runtime used for estimation increases linearly with the number of interventions, while the slope of the total runtime and the runtime used for training becomes steeper. This is because in each intervention,

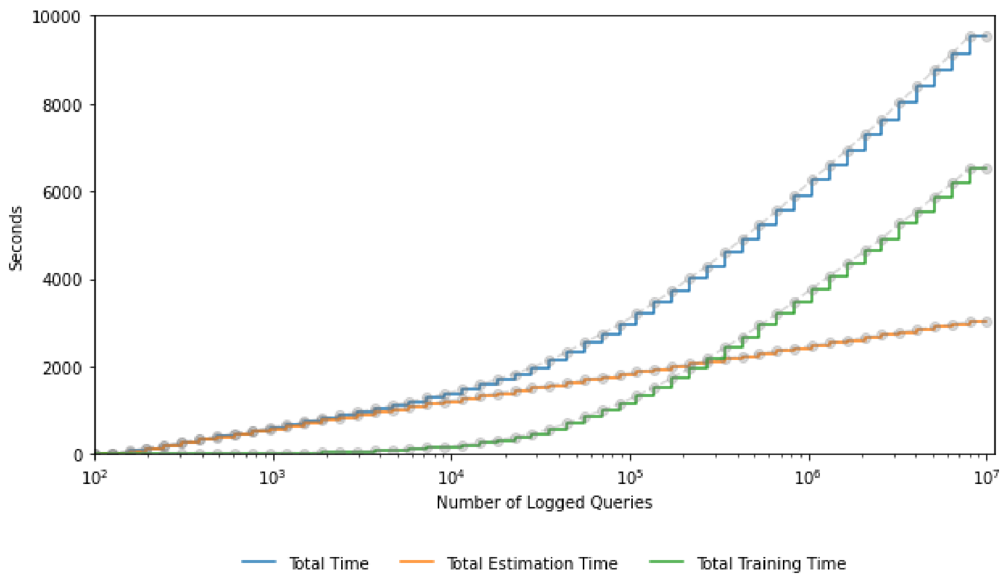

Figure 3: Runtime of the baseline model, the original intervention-aware estimator with 50 interventions and 1000 sampled rankings per query. Results based on an average of 20 runs.

the number of additional rankings generated for estimation remains constant, resulting in a consistent increase in estimation runtime. On the other hand, as the model becomes more complex and more click data is used for estimation, leading to more accurate estimates, the runtime for model training increases at a faster rate.

Overall, this figure provides valuable insights into the runtime of the entire optimization process, and give me a clear direction for the next step in improving the efficiency of the algorithm: saving time for bias estimation and model training. We can see that initially most of the time is spent on bias estimation, but later the training time overtakes it. Therefore, my focus will be on these two parts, as each is important at different stages of the learning process.

5.2 Bias Estimation Efficiency

To answer the Research Question 2: *which of the methods mentioned in Section 3.1, result in the shortest runtime for bias estimation while maintaining the performance*, I consider Figure 4 which displays how different numbers of sampled rankings and different kinds of estimation methods affect performance.

First, I consider the left column of Figure 4 which displays performance of different kinds of estimation methods, including Frequency Estimation, Plackett-Luce Estimation, Direct Estimation Using Displayed Rankings, Utilizing Displayed Rankings with Additional Rankings and Frequency Expectation Calculation. The three sub-figures from top to bottom represent the performance when $N = \{10, 100, 1000\}$, respectively. From the figure, it can be seen that, no matter what the value of N, when using Direct Estimation with Displayed Rankings (green line) and Utilizing Displayed Rankings with Additional Rankings (red line), there is a significant performance gap compared to the other three methods. These two methods have one thing in common, they both estimate bias based on (extended) collected interaction data, which lead to inaccurate estimations related to queries that have not been searched by users, resulting in low performance. We can

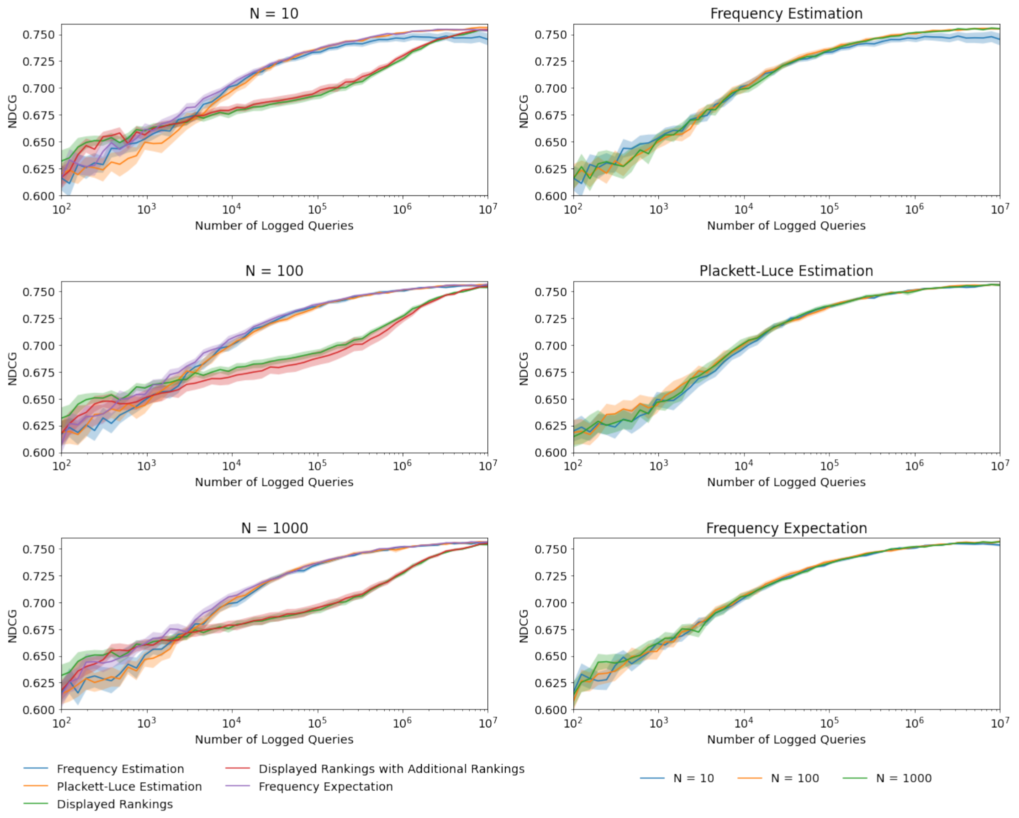

Figure 4: Comparison of different numbers of sampled rankings and different kinds of estimation methods. Left: the performance of different methods with the same number of sampled rankings; Right: the performance of the same method with different numbers of sampled rankings. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

conclude that neither method is suitable for improving the efficiency of estimation, as they cannot even guarantee performance.

I therefore turn to the right column of Figure 4 which shows the performance of the remaining three methods: Frequency Estimation, Plackett-Luce Estimation, and Frequency Expectation Calculation when estimated using three different numbers of rankings. We see that, except for the Frequency Estimation with $N = 10$, which have obvious performance loss, the remaining eight configurations all have almost the same performance curves. This means that a smaller number of additional rankings is sufficient to compute accurate estimates for all three methods.

Given that the eight configurations above maintain optimal performance, it is important to compare their runtime (both total time and total estimation time) to decide which configuration is most efficient. I consider Figure 5 which displays total time and Figure 6 which displays total estimation time of eight different configurations for bias estimation.

First, from the left column of Figure 5, it can be seen that when $N = 10$, the Plackett-Luce Estimation method takes the least time, about 6700 seconds; when $N = 100$, the Frequency Estimation method takes the least time, about 7100 seconds; when $N = 1000$, the Frequency Expectation Calculation method takes the least time, about 8400 seconds. This suggests that when N takes different values, the method that takes the least time is also different. Then look at the right column of Figure 5, we can observe that no matter what method is used, as the number of sampling rankings increases, the time will naturally become longer. In summary, when considering the shortest total time, the best configuration is the Plackett-Luce Estimation method with $N = 10$ additional sampled rankings per query for estimation.

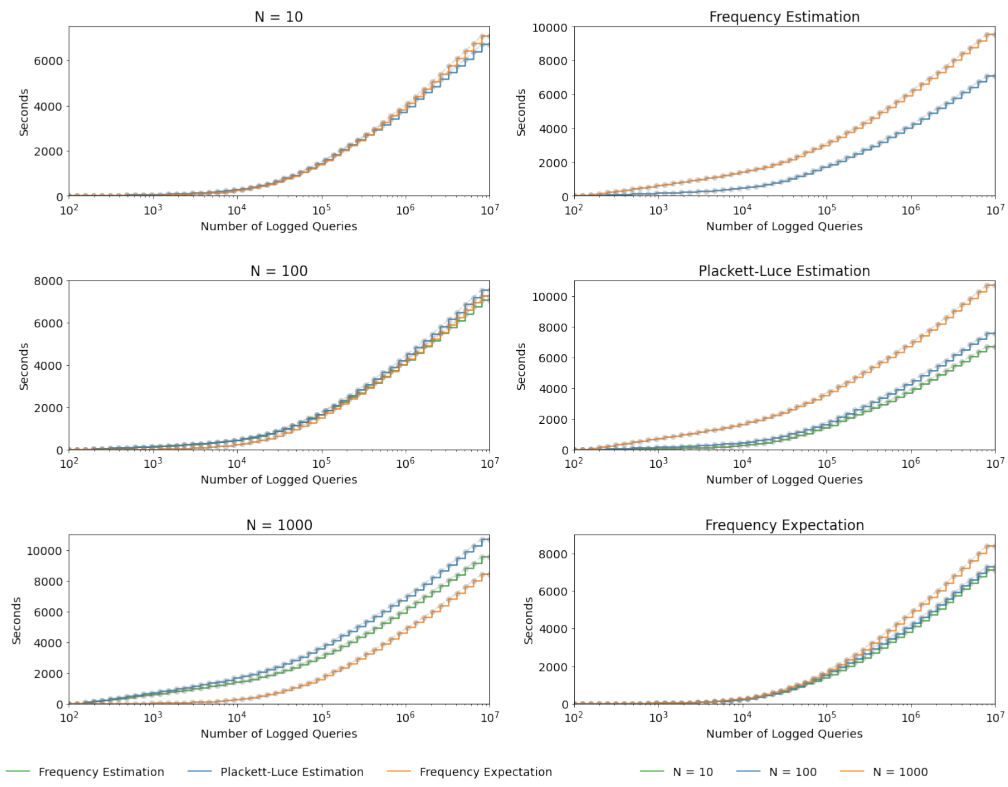

Figure 5: The **total time** of the remaining eight configurations which maintain optimal performance. Left: the total time of different methods with the same number of sampled rankings; Right: the total time of the same method with different numbers of sampled rankings. Results based on an average of 20 runs.

Second, from the left column of Figure 6, we can find that, unlike the left column of Figure 5 (total time), the Frequency Expectation Calculation method is always the least time-consuming for the total estimation time, and it is significantly lower than the other two. And when we look at the right column, we can see that fewer sampled rankings can indeed reduce the estimation time very significantly. So we can know from the sub-figure in the last row of the right column that, among all eight configurations, the Frequency Expectation Calculation method with $N = 10$ additional sampled rankings takes the shortest estimation time, which is about 120 seconds.

In summary, I can answer Research Question 2 and conclude that the configuration (the Plackett-Luce Estimation method with $N = 10$) that takes the shortest total time and the configuration (the Frequency Expectation Calculation method with $N = 10$) that takes the shortest estimation time, respectively. I keep both configurations and experiment separately. Particularly for the latter one, it has the shortest estimation time, but its total time is still higher than the other configuration. This indicates that it spends more time on model training, making it more necessary to optimize the model training process.

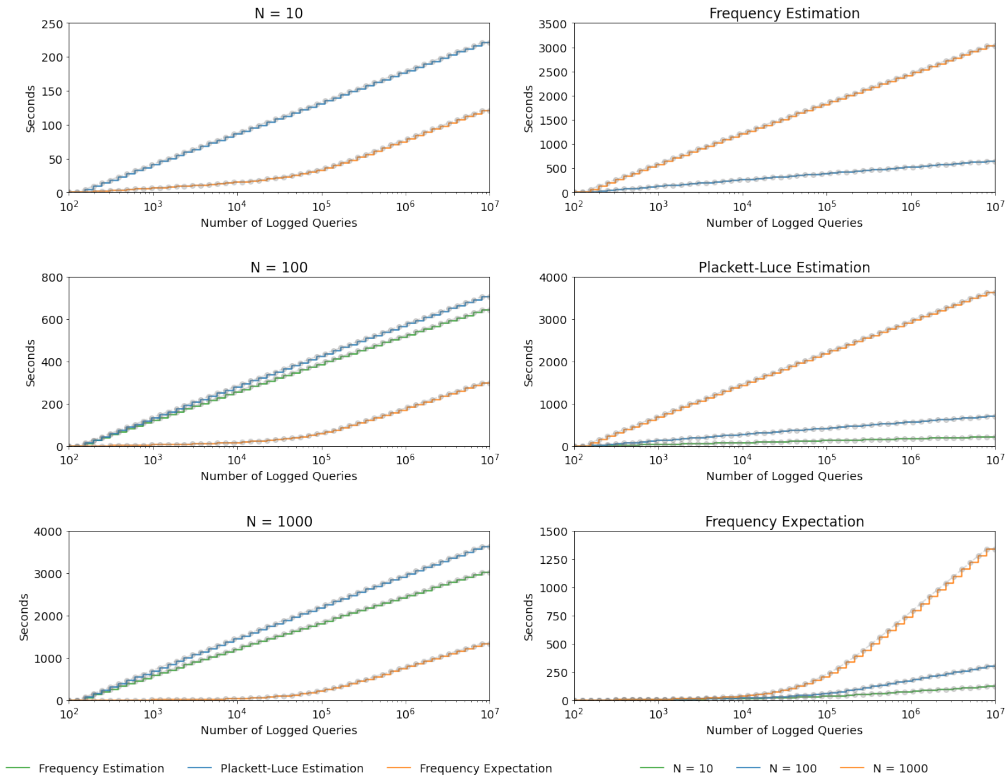

Figure 6: The **total estimation time** of the remaining eight configurations which maintain optimal performance. Left: the total estimation time of different methods with the same number of sampled rankings; Right: the total estimation time of the same method with different numbers of sampled rankings. Results based on an average of 20 runs.

5.3 Model Training Efficiency

To better understand why the maximum epoch should be set, and what the maximum epoch value should be, I first consider the figure 7, which displays the performance of the estimator and the number of epochs which is determined by the early stopping procedure required to complete model training in an intervention. This allows me to spot room for improvement in runtime.

We can observe that from $|\mathcal{D}| = 10^6$ to $|\mathcal{D}| = 10^7$, the performance of the estimator does not improve much, but the number of epochs required to complete the model training increases. So I set different maximum epoch values, both fixed and varying: $\text{max\_epochs} = \{6, \lfloor \log_{10}(|\mathcal{D}|) \rfloor - 1\}$, as shown by the dotted line in the figure, all model training stops when the dashed line is reached.

To answer the Research Question 3: *what methods, including setting different maximum epoch values, iterative training with and without weight decay, result in the most efficient model training process with maintained or improved accuracy of the results*, I consider Figure 8 which shows the effect of setting the maximum epoch and Figure 9 which shows the effect of utilizing different model initialization strategies.

It can be seen from Figure 8 that setting the maximum epoch can indeed reduce the running time while maintaining performance, However, depending on the specific maximum epoch value, the reduction time is also different. Comparing the two sub-

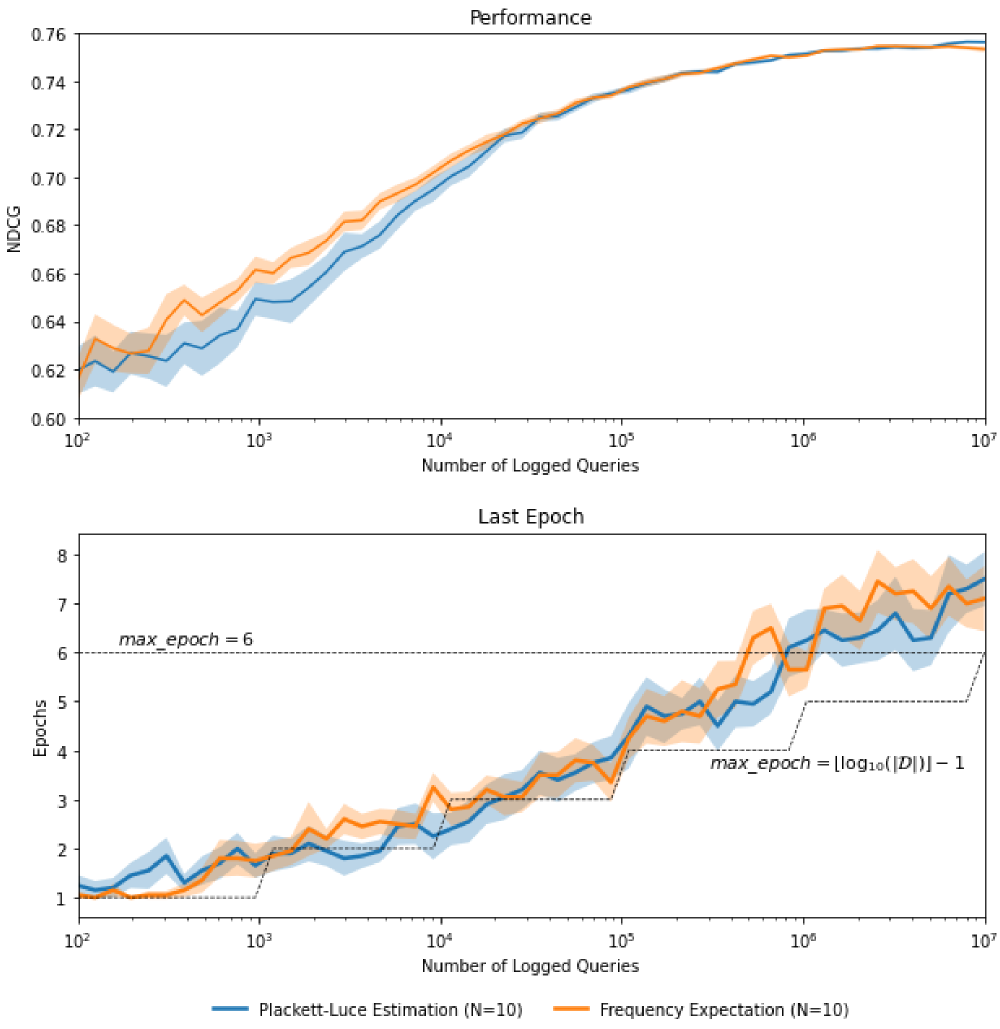

Figure 7: Comparison of two optimal configurations. Top: Performance of the two optimal configurations. Bottom: Last epoch for the two optimal configurations. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

figures in the last row of Figure 8, it can be concluded that max_epochs $= \lfloor \log_{10}(|\mathcal{D}|) \rfloor - 1$ saves significantly more time than max_epochs $= 6$. In addition, it should be noted that even if the training time is optimized by setting the maximum epoch, the configuration (the Frequency Expectation Calculation method with $N = 10$) still takes a little more time than the configuration (the Plackett-Luce Estimation method with $N = 10$).

I analyze Figure 9 in the same way, and from Figure 9 we can see that its situation is just the opposite of that of Figure 8. We can find that after applying iterative training, although the running time is greatly reduced, this has a very bad impact on performance. When we look at the left column, we can see that without weight decay, iterative training completely destroys the original performance, even though it significantly reduces the running time.

When we turn to the right column of Figure 9, we can observe that when we apply weight decay, the Plackett-Luce Estimation method regains its original performance, while the Frequency Expectation Calculation method performs just as poorly as when I

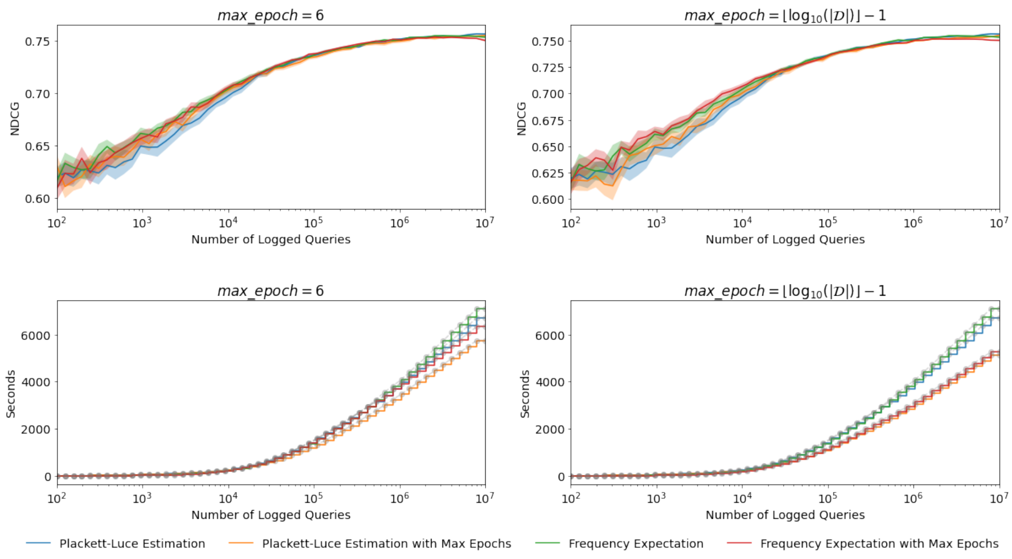

Figure 8: Performance and runtime effects of applying the maximum epoch. Left: max_epochs = 6. Right: max_epochs = $\lfloor \log_{10}(|\mathcal{D}|) \rfloor - 1$. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

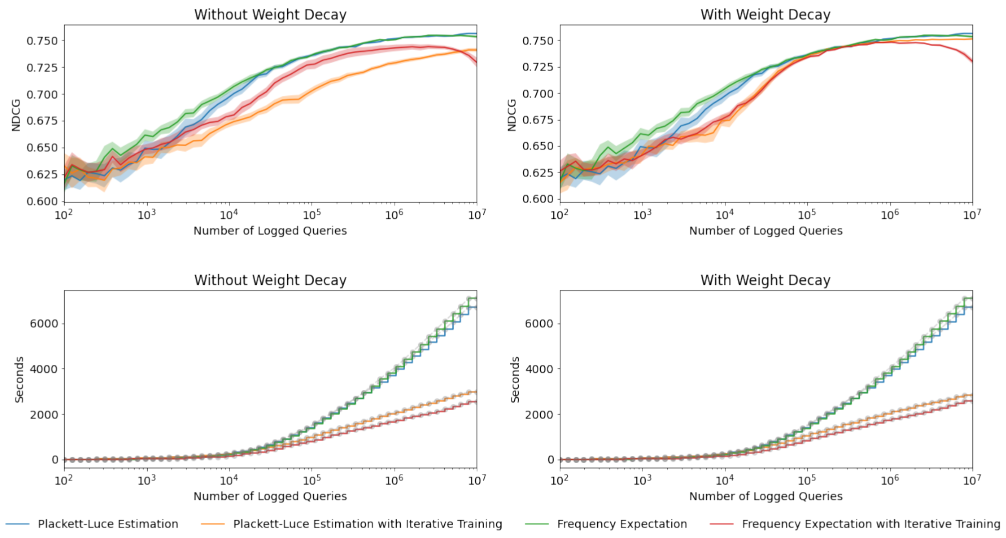

Figure 9: Performance and runtime effects of applying iterative training. Left: Without weight decay. Right: With weight decay. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

do not apply weight decay.

It can be seen from Figure 8 and Figure 9 that after optimizing the model training process, the configuration (the Plackett-Luce Estimation method with $N = 10$) is always better than the configuration (the Frequency Expectation Calculation method with $N = 10$), so next, I will only use the configuration (the Plackett-Luce Estimation method with $N = 10$) for experiments.

In the final analysis of Research Question 3, I will combine the two methods (Iterative

Training with Weight Decay and Scaling max_epoch) and compare its performance with the baseline model and the optimal models obtained from each previous analysis. I consider Figure 10, which shows the comparison between the optimal model obtained from each of my previous analysis and the baseline model.

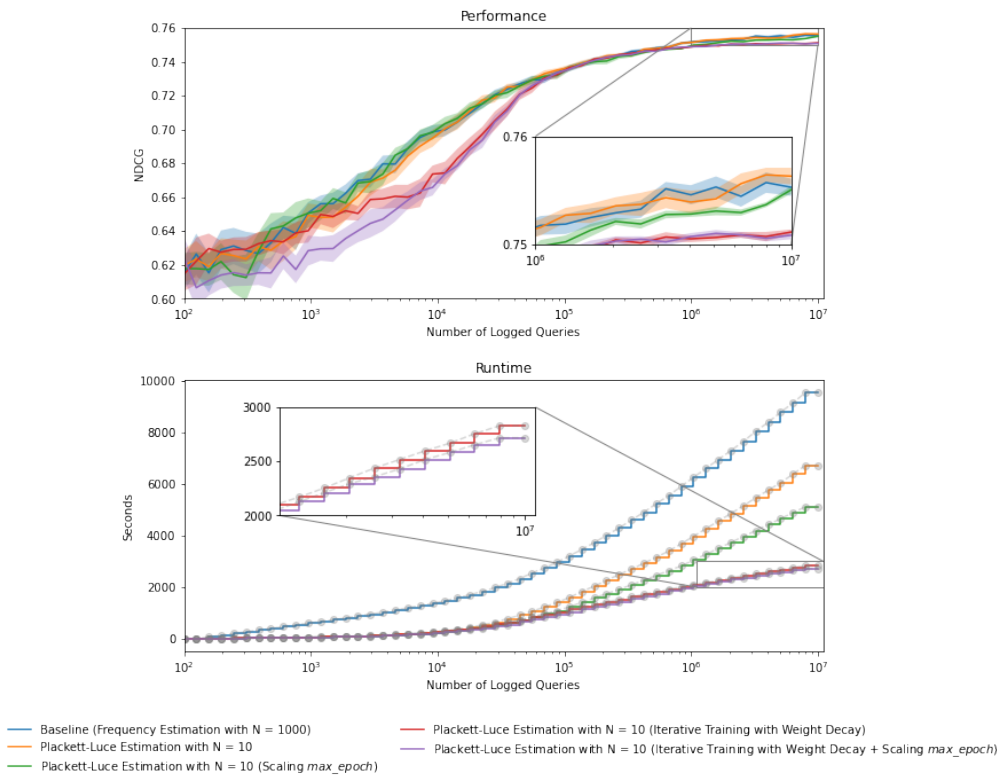

Figure 10: Comparison of 4 different optimal methods (from the previous analysis) and the baseline model. Top: Performance. Bottom: Runtime. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

From Figure 10, we can see that, compared with the **Baseline Estimator** (the original intervention-aware estimator using frequency estimation with $N = 1000$ sampled rankings), estimators applying iterative training have a certain loss in performance, and there is also a slight penalty in performance for estimator applying the maximum epoch, while the estimator without any optimization of the training process maintains the performance. All of them reduce the running time to a certain extent.

Putting it all together, I answer the Research Question 3 positively: the **Fastest Estimator** involves utilizing $N = 10$ additional sampled rankings for Plackett-Luce Estimation and entails employing scaling max epochs and iterative training with weight decay. Moreover, for optimal performance with minimal runtime, the overall **Best Estimator** is to employ only the Plackett-Luce Estimation alongside $N = 10$ additional sampled rankings, without any optimization of the model training process.

5.4 Real-Time Interactive Simulation

To answer the Research Question 4: *does the reduced running time of the intervention-aware approach result in an meaningful difference in responsiveness*, I consider Figure

11 which displays the performance and the number of interventions of three different estimators in interactive systems with different user sizes during one-hour simulation.

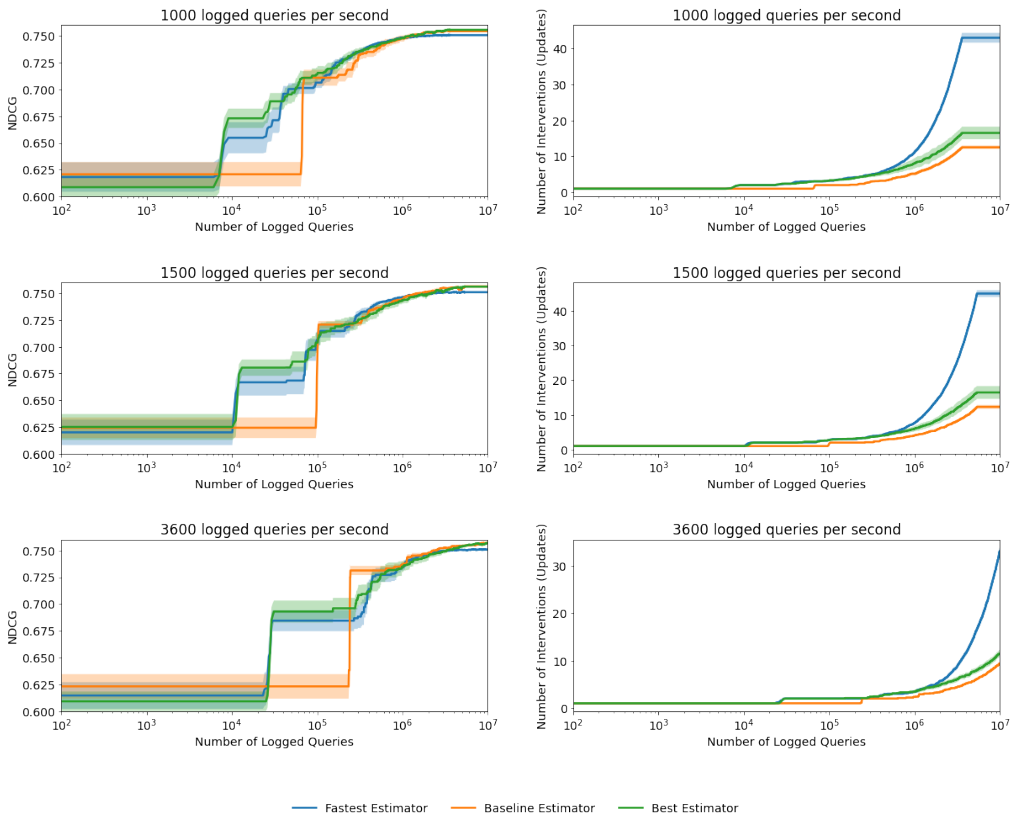

Figure 11: Comparison of **Baseline Estimator**, **Fastest Estimator** and **Best Estimator** in user interaction systems with different user sizes (the number of logged queries generated per second is different) during one-hour simulation. Left: Performance. Right: Number of interventions. Results based on an average of 20 runs, shaded area indicates the 90% confidence intervals.

It can be clearly observed from the left column of Figure 11 that in the initial stage of the user-system interaction I simulated, both **Fastest Estimator** and **Best Estimator** are significantly better than the **Baseline Estimator**. This is due to the faster intervention of **Fastest Estimator** and **Best Estimator**, that is to say, less time is required to complete a logging policy update. As the number of logged queries increases, the performance of the three estimators gradually tends to be consistent, and finally converges. It can be clearly seen that there is still a small gap between the **Fastest Estimator** and the **Baseline Estimator**, while **Best Estimator** ends up slightly outperforming the other two estimators. Because the analysis in Section 5.3 concludes that **Best Estimator** not only greatly reduces the running time, allowing more interventions, but also maintains the optimal performance.

At the same time, comparing the number of interventions of the three different estimators in the right column of Figure 11, we can find that the **Fastest Estimator** has the most number of interventions and is significantly ahead of the other two estimators. Although this makes the **Fastest Estimator** achieve fairly good performance in the initial stage of the simulation, it is consistently the worst in the final performance. The reason is that when more and more data are available and the model becomes more and more complex, too frequent interventions (updates) will cause the model to fail to learn complete information from the data.

Thus, I answer the Research Question 4 positively: the running time saved by **Best Estimator** which only optimizes the bias estimation but not the model training does translate into the responsiveness of the interactive system, achieving faster and better performance.

6 CONCLUSION

6.1 Summary

In conclusion, this thesis builds upon the intervention-aware estimator, a previously proposed approach for learning from user interactions, and explores the potential of making this estimator more efficient. The main focus of this thesis is to optimize the method to reduce computational load and overall runtime without compromising performance.

In response to Research Question 1, the study identifies bias estimation and model training as critical time-consuming stages and highlights the need for enhancing their efficiency. Addressing Research Question 2, the Plackett-Luce Estimation method with $N = 10$ additional sampled rankings is found to take the shortest total time, while the Frequency Expectation Calculation method with $N = 10$ additional sampled rankings demonstrates the shortest total estimation time. These findings prompt further investigation into optimizing model training. Research Question 3 concludes that the **Fastest Estimator** involves utilizing $N = 10$ additional sampled rankings for Plackett-Luce Estimation and entails employing scaling max epochs and iterative training with weight decay. However, for optimal performance with minimal runtime, the **Best Estimator** suggests using only Plackett-Luce Estimation with $N = 10$ rankings. Lastly, Research Question 4 establishes that the saved runtime through the **Best Estimator** really helps to improve the responsiveness of the interactive system.

By addressing these research questions, this thesis aims to contribute to the development of fast and accurate intervention-aware counterfactual estimators for online learning to rank. I finally conclude that the time saved by faster bias estimation and model training can lead to more interventions, and more interventions can indeed bring higher performance to a certain extent, but too frequent intervention within a certain period of time can degrade performance. The best strategy to achieve a faster and better performance thus a more responsive user experience with the system is to use faster bias estimation methods combined with more adequate and complete but relatively slower model training methods for online learning to rank with intervention-aware counterfactual estimators.

6.2 Future Work

First, future work could extend the analysis to other publicly available LTR industry datasets, such as MSLR-WEB30k [24] and Istella LETOR [7] . Second, future focus may be on scenarios where α and β are not given and need to be estimated by expectation-maximization (EM) methods. Additionally, future research will explore the potential of applying double robust (DR) estimation for click data [19].

References

[1] Aman Agarwal, Xuanhui Wang, Cheng Li, Michael Bendersky, and Marc Najork. Addressing trust bias for unbiased learning-to-rank. In *The World Wide Web Conference*, WWW '19, page 4–14, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. Proceedings of the Seventh International World Wide Web Conference.

[3] Sebastian Bruch, Shuguang Han, Michael Bendersky, and Marc Najork. A stochastic treatment of learning to rank scoring functions. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 61–69, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors, *Proceedings of the Learning to Rank Challenge*, volume 14 of *Proceedings of Machine Learning Research*, pages 1–24, Haifa, Israel, 25 Jun 2011. PMLR.

[5] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. *Click Models for Web Search.* Morgan & Claypool, 2015.

[6] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, page 87–94, New York, NY, USA, 2008. Association for Computing Machinery.

[7] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Trans. Inf. Syst.*, 35(2), dec 2016.

[8] Fernando Diaz, Bhaskar Mitra, Michael D. Ekstrand, Asia J. Biega, and Ben Carterette. Evaluating stochastic rankings with expected exposure. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 275–284, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Rolf Jagerman, Harrie Oosterhuis, and Maarten de Rijke. To model or to intervene: A comparison of counterfactual and online learning to rank from user interactions. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 15–24, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, page 154–161, New York, NY, USA, 2005. Association for Computing Machinery.

[11] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2):7–es, apr 2007.

[12] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 781–789, New York, NY, USA, 2017. Association for Computing Machinery.

[13] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[14] R.D. Luce. *Individual Choice Behavior: A Theoretical Analysis.* Dover Books on Mathematics. Dover Publications, 2012.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[16] Harrie Oosterhuis. Learning from user interactions with rankings : a unification of the field, 2020.

[17] Harrie Oosterhuis. Computationally efficient optimization of plackett-luce ranking models for relevance and fairness. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'21).* ACM, 2021.

[18] Harrie Oosterhuis. Learning-to-rank at the speed of sampling: Plackett-luce gradient estimation with minimal computational complexity. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2266–2271, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Harrie Oosterhuis. Doubly robust estimation for correcting position bias in click feedback for unbiased learning to rank. *ACM Trans. Inf. Syst.*, 41(3), feb 2023.

[20] Harrie Oosterhuis and Maarten de Rijke. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1293–1302, New York, NY, USA, 2018. Association for Computing Machinery.

[21] Harrie Oosterhuis and Maarten de Rijke. Policy-aware unbiased learning to rank for top-k rankings. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 489–498, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Harrie Oosterhuis and Maarten de Rijke. Unifying online and counterfactual learning to rank: A novel counterfactual estimator that effectively utilizes online interventions. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 463–471, New York, NY, USA, 2021. Association for Computing Machinery.

[23] R. L. Plackett. The Analysis of Permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 12 2018.

[24] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets, 2013.

[25] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[26] Yuta Saito, Aihara Shunsuke, Matsutani Megumi, and Narita Yusuke. Open bandit dataset and pipeline: Towards realistic and reproducible off-policy evaluation. *arXiv preprint arXiv:2008.07146*, 2020.

[27] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, nov 1975.

[28] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986.

[29] Mark Sanderson. Test collection based evaluation of information retrieval systems. *Foundations and Trends® in Information Retrieval*, 4(4):247–375, 2010.

[30] K. Sparck Jones, S. Walker, and S.E. Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 1. *Information Processing & Management*, 36(6):779–808, 2000.

[31] K Sparck Jones, S Walker, and S.E Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information Processing & Management*, 36(6):809–840, 2000.

[32] Niek Tax, Sander Bockting, and Djoerd Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51(6):757–772, 2015.

[33] Ali Vardasbi, Harrie Oosterhuis, and Maarten de Rijke. When inverse propensity scoring does not work: Affine corrections for unbiased learning to rank. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 1475–1484, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 115–124, New York, NY, USA, 2016. Association for Computing Machinery.

[35] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1313–1322, New York, NY, USA, 2018. Association for Computing Machinery.

[36] Wikipedia contributors. Online machine learning — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Online_machine_learning&oldid=1168656498`, 2023. [Online; accessed 20-August-2023].

[37] Wikipedia contributors. Ranking (information retrieval) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Ranking_(information_retrieval)&oldid=1149383161`, 2023. [Online; accessed 8-August-2023].

[38] Yisong Yue and Thorsten Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1201–1208, New York, NY, USA, 2009. Association for Computing Machinery.