Efficient and Effective Tail Latency Minimization in Multi-Stage Retrieval Systems

Joel Mackenzie
RMIT University
Melbourne, Australia
joel.mackenzie@rmit.edu.au

J. Shane Culpepper
RMIT University
Melbourne, Australia
shane.culpepper@rmit.edu.au

Roi Blanco
RMIT University
Melbourne, Australia
rblanco@udc.es

Matt Crane
University of Waterloo
Waterloo, Canada
matt.crane@uwaterloo.ca

Charles L. A. Clarke
University of Waterloo
Waterloo, Canada
claclarke@gmail.com

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

ABSTRACT

Scalable web search systems typically employ multi-stage retrieval architectures, where an initial stage generates a set of candidate documents that are then pruned and re-ranked. Since subsequent stages typically exploit a multitude of features of varying costs using machine-learned models, reducing the number of documents that are considered at each stage improves latency. In this work, we propose and validate a unified framework that can be used to predict a wide range of performance-sensitive parameters which minimize effectiveness loss, while simultaneously minimizing query latency, across all stages of a multi-stage search architecture. Furthermore, our framework can be easily applied in large-scale IR systems, can be trained without explicitly requiring relevance judgments, and can target a variety of different efficiency-effectiveness trade-offs, making it well suited to a wide range of search scenarios. Our results show that we can reliably predict a number of different parameters on a per-query basis, while simultaneously detecting and minimizing the likelihood of tail-latency queries that exceed a pre-specified performance budget. As a proof of concept, we use the prediction framework to help alleviate the problem of tail-latency queries in early stage retrieval. On the standard ClueWeb09B collection and 31k queries, we show that our new hybrid system can reliably achieve a maximum query time of 200 ms with a 99.99% response time guarantee without a significant loss in overall effectiveness. The solutions presented are practical, and can easily be used in large-scale distributed search engine deployments with a small amount of additional overhead.

CCS CONCEPTS

•Information systems → Retrieval efficiency; Search engine architectures and scalability; Learning to rank;

KEYWORDS

Multi-Stage Retrieval; Query Prediction; Experimentation; Measurement; Performance

1 INTRODUCTION

The competing goals of maximizing both efficiency and effectiveness in large-scale retrieval systems continue to challenge builders of search systems as the emphasis in modern architectures evolves towards multi-stage retrieval [43]. Many old efficiency problems become new again in the increasingly complex cascade of document re-ranking algorithms being developed. For example, research groups can focus on early stage retrieval efficiency [4, 16, 57], balancing feature costs [56, 59], or improving the performance of the learning-to-rank algorithms [5, 26, 36, 37, 39].

While great strides have been made in all of these areas, gaps remain in our understanding of the delicate balance between efficiency and effectiveness in each "stage" of the re-ranking cascade. One of the most significant limitations preventing further progress is in training data availability. While query sets to measure efficiency in various collections are plentiful, the costs of gathering relevance judgments in order to measure effectiveness limit the number of topics available for more detailed trade-off analyses.

In this work we explore how to apply a *reference list* framework [13, 48, 50, 58] to alleviate this problem. We leverage the new framework to build machine-learned models capable of predicting query response times, candidate set sizes in early stage retrieval, and algorithm aggressiveness to balance efficiency and effectiveness on a query-by-query basis. In particular, we focus on using this unified framework to identify and reduce *tail-latency queries* [18, 24, 25, 28], i.e., those with unusually large response time. We explore three important research questions:

Research Question 1 (RQ1): *What is the best way to use reference lists to accurately perform dynamic per query parameter predictions in early stage retrieval?*

Research Question 2 (RQ2): *What is the relationship between tail-latencies and index traversal algorithm, and can our new prediction framework be used reliably provide worst case guarantees on first-stage query efficiency?*

Research Question 3 (RQ3): *What combination of predictions will lead to efficient first-stage retrieval, minimizing the number of candidate documents exiting the first stage (and thus making later stages more efficient), and also minimize effectiveness loss in final stage re-ranking?*

In answering these questions, our research contributions include:

(1) A unified framework that can be used to predict a wide variety of performance-sensitive parameters in multi-stage retrieval systems.

(2) A pragmatic, yet highly effective solution to tail-latency query minimization that can easily be implemented in large-scale retrieval systems, and provide worst case performance guarantees on performance.
(3) A pathway to more fine-tuned per-query optimization techniques, and the tools necessary to implement and test systems leveraging these ideas.

We achieve these goals using three ideas. First, we exploit the idea of query difficulty prediction [10] and static pre-retrieval features to build a unified prediction framework. Next, we explore the relationship between the number of documents returned in a top-k candidate set and the index traversal algorithm. Three different index traversal algorithms have been commonly used: document-at-a-time (DaaT), term-at-a-time (TaaT), and score-at-a-time (SaaT). A recent paper by Crane et al. [14] performed a comprehensive comparison of state-of-the-art DaaT and SaaT algorithms and found that both approaches have advantages and disadvantages. In this work we look at a simple index mirroring approach which selectively uses the best algorithm based on a series of pre-retrieval predictions. Finally, the efficiency predictors are integrated with an effectiveness loss minimization prediction. Together, this series of "Stage-0" pre-retrieval predictions produces a pipeline that maximizes efficiency and effectiveness in a multi-stage retrieval system, and is capable of achieving 99.99% response time guarantees when using a worst case running time of 200 ms on a commonly used web collection.

2 BACKGROUND AND RELATED WORK

Efficient Query Processing. Efficient query processing can be attained through a range of index organizations and traversal strategies based on the *inverted index* data structure [62]. Document-at-a-time (DaaT) query processing relies on postings lists being sorted in ascending order of the document identifiers. At query time, a pointer is set at the beginning of each postings list. Once the current document has been evaluated, the pointers are forwarded to the next document in the lists. An efficient method for *disjunctive* DaaT processing is the *Weak-AND* (Wand) algorithm [8]. In order to support Wand traversal, the upper-bound score that term t can contribute to any given document must be pre-computed and stored in the index (U_t). At query time, Wand uses the lowest-scoring heap document as a *threshold*. When selecting the next document in which to score, Wand will only select a document in which the sum of the U_t scores is larger than the heap threshold. The advantage of Wand is that documents that are not able to make the final top-k results are able to be safely ignored, making it highly efficient. Although originally aimed for traversing on-disk indexes, Wand has been proven to be efficient in-memory on many occasions [4, 14, 22, 41, 44, 51].

Ding and Suel [21] (and at a similar time, Chakrabarti et al. [11]) explored an improved version of Wand named *Block-Max WAND* (Bmw). The key observation in Bmw is that since many index compression algorithms are block-based [32, 61], skipping can be achieved at the block level, thus saving an entire block decompression. In order the facilitate this skipping, the U_t score is computed for every *block* in each postings list, known as the $U_{b,t}$ score. When a pivot document is found (by summing the U_t scores until the threshold is exceeded), the local block score is then used to refine the estimated score, that is, the sum of the $U_{b,t}$ scores is computed. If this sum still exceeds the threshold, then the pivot document is scored. Otherwise, a new pivot is selected. Additional gains from Bmw are achieved through an improved skipping function that identifies if the current block configuration could not contain a document with a score above the threshold. If this condition is met, a new pivot is selected that *may* contain enough weight to enter the top-k heap. Further enhancements to Bmw have been made in the literature, usually by using additional auxiliary structures that provide a quicker search time while using additional space, or using hybrid indexes [19, 45, 46].

Another entirely different method for top-k query processing is the term-at-a-time (TaaT) and the closely related score-at-a-time (SaaT) approach. Term-at-a-time processing opts to process an entire postings list before moving onto the next list. Clearly, an additional data structure must be kept to store the partially accumulated scores while processing the lists. Anh et al. [2] made the observation that the term weight for any given document $w_{d,t}$ could be pre-computed and stored, rather than the term frequencies ($f_{d,t}$). Since the $w_{d,t}$ are typically floating point numbers, they are quantized into integer values to facilitate compression [2], the range of which impacts both effectiveness and efficiency [15]. For score-at-a-time processing, each postings list is sorted by *decreasing* impact score, which allows the most high scoring documents for each term to be processed first, and can allow for early-termination without sacrificing effectiveness. Recently, Lin and Trotman [34] introduced Jass, a modern SaaT algorithm which can be used for *anytime* retrieval, making it suitable for use in time-constrained environments and for controlling tail latencies.

Finally, some optimizations can be generalized to all index structures. For example, many compression algorithms have been proposed in the literature [32, 53, 55, 61] which are often applicable to frequencies, (quantized) document weights, and DocIDs. Another general improvement is to apply a special ordering to the DocID space [20, 27, 51]. Assignment strategies such as lexicographically sorting the DocIDs by the corresponding *URL* has been shown to improve both the compression rate, and reduce the query latency [49, 51].

Tail Latencies. A tail-latency query is an "outlier" query whose response time occurs above the nth percentile, where n is a large value such as 95, 99, or even 99.99 [28, 60]. As collections grow larger, systems must scale accordingly. As systems become more complex, the probability of tail latencies occurring also increases [18], particularly for distributed architecture where end-to-end latency is often bound by the slowest component. Tail latencies can be addressed through either hardware or software optimizations, or both. For example, replicating and partitioning collections [18, 23, 29, 30] allows effective load balancing which can minimize tail-latency queries.

Previous work has attempted to reduce tail latencies in a range of different contexts. Jeon et al. [25] focus on 99th percentile tail-latency queries at the level of a single *Index Server Node* (ISN) by predicting long running queries, and running them in parallel. Queries that are *not* predicted as long running are simply ran

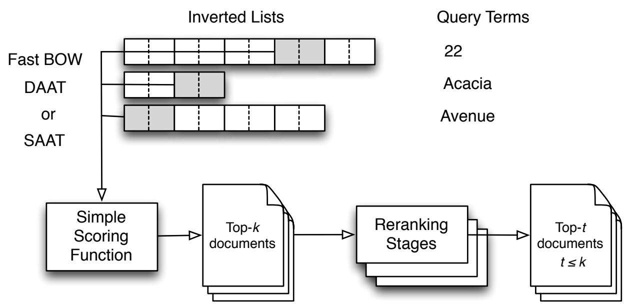

Figure 1: Architecture of a typical multi-stage retrieval system. Queries are first processed using an efficient bag-of-words processing algorithm. The initial candidate set of k documents then undergoes a series of re-ranking stages where the candidate pool is shrunk, and more expensive learning-to-rank algorithms are used to produce a final set of top-t documents to return to the user, where often $t \ll k$.

sequentially, which avoids the overhead cost of parallalization. Another recent work targets extreme tail latencies (ie, at the 99.99th percentile) [24, 28]. This target is achieved through *Dynamic, Delayed, Selective* (DDS) prediction. DDS prediction works as follows. First, a new query is ran for a short time, such as 20ms, and dynamic features are collected from this initial processing. Then, new dynamic features (and, some additional static features) are used to predict whether the query is a long running query. If so, or if there is reasonable uncertainty (based on the predicted error), then the query will be accelerated using parallelization. The prediction error is then used to improve coverage of midpredicted true long running queries.

Beyond the tail latency in ISNs, DDS also reduces the latency of the aggregator node, which aggregates the results from the multiple ISNs before reporting them to the user. Yun et al. [60] also address the problem of aggregating information from ISNs, but this is orthogonal to our work, which focuses on the processing at an ISN, and not at the aggregation node.

Multi-stage Search Architectures. Multi-stage retrieval has become the dominant model in modern web search systems [3, 4, 9, 38, 39, 43]. In this approach, a set of candidate documents are generated that are likely to be relevant to a query, and then in one or more stages, the document sample is iteratively reduced and reordered using a series of increasingly expensive machine learning techniques. Since re-ordering can be computationally expensive and is sensitive to the number of documents that must be reordered, minimizing the size of the candidate set is an important problem [9, 38, 52].

Figure 1 exemplifies a typical multi-stage retrieval architecture. A fast bag-of-words processing algorithm produces a top-k candidate set. This initial set of documents is then re-ranked one or more times using a learning-to-rank algorithm to produce a final output set of t documents, where $t \leq k$, and can be $t \ll k$ in some configurations.

Efficiency matters at all stages of the process. Kohavi et al. [31] showed that every 100 ms boost in overall search speed increases revenue by 0.6% at Bing. So, even small gains in overall performance can translate to tangible benefits in commercial search engines. Efficiency remain an important problem in multi-stage retrieval with papers focused on cascaded ranking [43, 59], and early exit optimizations [9, 17]. Recently, Wang et al. [57] proposed a fast candidate generation framework which opts to build a two-layer index. The bottom layer is the standard inverted index, and the top layer is a single or dual-term auxillary structure which stores a subset of the bottom layer documents, sorted by impact score. At query time, a prefix of the top layer is accessed, which generates a set of candidate documents. Then, the most promising of these candidate documents has its partial scores updated by accessing the lower layer of the index (to achieve a more accurate score). Finally, the top-c candidates are selected and passed onto the next stage of the cascade. We do not consider this generation framework as it provides approximate results, but note that it can be directly applied to our existing Bmw ISN to improve efficiency (with some small loss in effectiveness). We leave this as future work.

Effectiveness Evaluation in Multi-Stage Retrieval. One obvious question arises when trying to measure trade-offs in multi-stage retrieval systems. The simplest approach is to simply make changes to the system, and re-compute a standard information retrieval metric such as average precision (AP), expected reciprocal rank (ERR), normalized discounted cummulative gain (NDCG), or rank biased precision (RBP) on the last stage result [12, 42]. However, this is unwieldy in practice, as it can be very difficult to identify exactly what changes are resulting in effectiveness differences.

A better approach is to compute intermediate results at different stages of re-ranking, and measure the differences between the two. For example, in a simple two-stage system, we could generate the top-k list for both stages and somehow measure the similarity or difference between the two runs. We refer to this as a *reference list* comparison. For example, we could just compute the *overlap* between the two lists, and this methodology is still commonly used in recent work [57]. But in practice, this approach does not properly capture importance of rank position in the two lists. To alleviate this problem, Webber et al. [58] proposed rank-biased overlap (RBO). This is a non-conjoint list comparison metric that places more importance on the loss of higher ranking items in a list than lower ranking ones.

The goals of RBO were taken one step further by Tan and Clarke [50] in the metric *Maximized Effectiveness Difference* (MED) where the exact gain function used to compute the difference can depend on any utility-based evaluation metric, such as ERR, DCG, or RBP. Furthermore, MED has the additional advantage that if partial judgments are available for any of the queries, the information can be used directly for the final comparison. Informally, MED answers the following question: given an effectiveness metric and two ranked lists, $\mathcal{D}^a$ and $\mathcal{D}^b$, what is the maximum difference in the effectiveness scores between the two lists? Tan and Clarke [50] define variants of MED for many standard retrieval metrics, including average precision (MED-AP), expected reciprocal rank (MED-ERR), normalized discounted cumulative gain (MED-NDCG), and rank biased precision (MED-RBP). We refer the reader to the work of Tan and Clarke for the formal definition of MED. In this paper we employ MED-RBP with a decay value of 0.95 (MED-RBP$_{0.95}$) as our primary difference measure.

Other approaches to defining reference lists have been studied recently by Shtok et al. [48]. Their approach is orthogonal to the one taken in this work. The relationship between how best to construct ground truth runs and measure the similarity between two non-conjoint lists remains a fruitful area of future research in the IR community, but is beyond the scope of this work.

3 METHODOLOGY

In order to build our prediction framework, we need to account for several issues. First, we need a ground truth which represents an idealized last stage run over a large corpus of queries. This idealized last stage represents the reference list for which all comparisons can be made. In order to build a plausible reference list, we adopt the methodology of Clarke et al. [13]. The 2009 Million Query Track (MQ2009) query set was used to perform both training and testing. We filtered this query set by removing single term queries (which can be answered extremely efficiently by taking the first k documents from the relevant postings list of the impact-ordered ISN). Following Clarke et al., we use the `uogTRMQdph40` run as a reference list, as it was one of the highest performing runs across the evaluated query set, and had results for all of the queries in the collection. In addition, we filtered out 905 queries which reported a MED-RBP$_{0.95}$ score greater than 0.5 when applying the fixed-k early stage (with $k = 10{,}000$), as these results show a clear mismatch between the early and late stages we are presenting. After filtering, we retain a set of 31,642 MQ2009 queries. The first 50 queries are held out for final effectiveness validation since these queries correspond to the queries in the 2009 TREC Web Track, and a full set of relevance judgments are available. For all predictions, queries were randomly assigned to 10 folds, and standard 10 fold cross validation was performed to produce the query predictions.

We use only MED-RBP$_{0.95}$ with a small target threshold of $\epsilon = 0.001$ for all experiments as we wish to aggressively minimize effectiveness loss. Clarke et al. showed that other common utility-based metrics could also easily be used such as ERR and DCG, and achieve similar results in their experiments, but we do not explore that option in this work.

Experimental Setup. All experiments were executed on an idle 24-core Intel Xeon E5-2690 with 512 GB of RAM hosting RedHat RHEL v7.2. ATIRE [54] was used to parse and index the ClueWeb09B collection, which was stopped using the default Indri stoplist, and stemmed using an *s*-stemmer. Timings were conducted on an appropriate Bmw[1] or Jass[2] index, which use QMX compression [53, 55] and the BM25 scoring model. Each query is processed 5 times, and the average of the 5 runs is reported.

Prediction Framework. Recently, Culpepper et al. [16] described an effective approach to dynamically predicting k while minimizing the effectiveness loss. Their key idea was to use the reference list methodology described above to build ground truth labels to train a classifier. However, their approach has a few drawbacks. First, the cascade classifier they described is interesting but unconventional in that it requires multiple predictions to be made, depending on the final k. Fewer predictions are required for small k, but up to 8 independent predictions are required for large k. Secondly, the problem they describe is really a regression problem in practice. Using regression allows an exact k to be predicted instead of an approximate cutoff, which translates into fewer documents being re-ranked in later stages of the retrieval system.

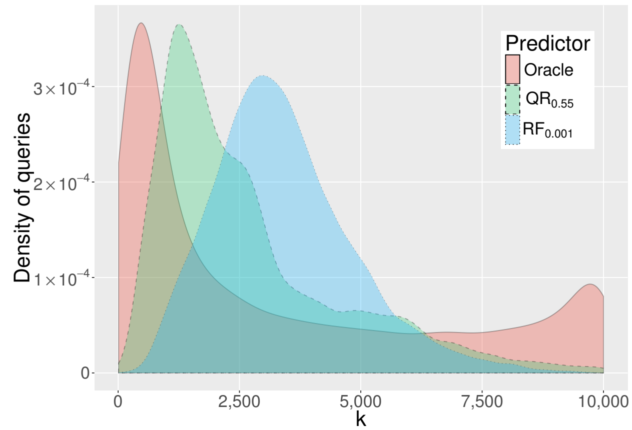

Figure 2: A comparison of the distributions of actual k versus predicted k when using a Random Forest regression and a Quantile Regression in first stage retrieval for the 31,642 queries from the MQ2009 TREC Task. Note that the Random Forest uses a training value of $\epsilon = 0.001$, whereas the best-fit distribution for the Quantile Regression was $\tau = 0.55$ for k.

Commonly, regression methods estimate the conditional expectation of a target dependent variable y given the independent variables (or features) $\mathbf{x}$. This implies that the method approximates the average value of the dependent variable when the independent variables are fixed. Given training data of the form $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)$ methods based on least squares try to optimize the loss function $L(\mathbf{x}, y) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(\mathbf{x_i} - y_i)^2$, which results in a good estimator for the mean $E[y|\mathbf{x}]$.

So, the obvious way to reproduce their work is to use a similar feature set, and compute the exact k needed for each query that achieves a very small expected MED loss, say, $\epsilon < 0.001$, and use a random forest to produce the predictions. When we build this training set, one immediate problem becomes apparent – the ground truth labels do not follow a standard distribution, but an out-of-the-box regression algorithm *does*. Figure 2 shows three different distributions – the true distribution of k in the ground truth set (Oracle), the random forest prediction (RF$_{0.001}$), and a quantile regression prediction (QR$_\tau$), which is described now.

A pitfall of standard regression methods is that they may become unstable under heavy-tailed distributions due to the dominant effects of outliers, or more precisely, when samples from the tail of the distribution have a strong influence on the mean. How to cope with this problem has been studied in the context of *robust estimation.* These estimators embody a family of methods designed to be more resilient to the data generation process by not following

[1] http://github.com/JMMackenzie/Quant-BM-WAND
[2] http://github.com/lintool/JASS/

the underlying assumptions behind the regressor; in the context of least squares, this would be errors being uncorrelated and having the same variance.

One simple way of dealing with the outlier problem is *quantile regression* which estimates either the conditional median or other quantiles of the response variable. If y has a cumulative distribution of $F_y(z) = p(y \leq z)$ then the τ-th quantile of y is given by $Q_y(\tau) = F_y^{-1} = \inf\{z : F_y(z) \geq \tau\}$. To learn a regressor that minimizes a τ value, we define the loss function $\xi_\tau(y) = y(\tau - \mathcal{I}\{y < 0\})$ where $\mathcal{I}\{\cdot\}$ is the indicator function. Therefore, τ-th quantile regression estimates the conditional τ-th quantile $F^{-1}_{y|\mathbf{x}}(\tau)$, or we want an estimate $\hat{f}_\tau$ such that $p(y < \hat{f}_\tau(\mathbf{x})) = \tau$:

$$\hat{f}_\tau = \operatorname*{argmin}_{f \in \mathcal{F}_\tau} \sum_{i=1}^{n} \xi_\tau(y_i - f(\mathbf{x_i})) = \quad (1)$$

$$\operatorname*{argmin}_{f \in \mathcal{F}_\tau} \left[(1-\tau) \sum_{y_i < f(\mathbf{x_i})} |y_i - f(\mathbf{x_i})| + \tau \sum_{y_i \geq f(\mathbf{x_i})} |y_i - f(\mathbf{x_i})| \right], \quad (2)$$

where $\mathcal{F}_\tau$ is a predetermined class of functions.

A robust regression method is *random forests* (RF) which build several decision trees using attribute bagging. In a nutshell, the algorithm samples with replacement the training data B times and trains several decision trees f_b using only each portion of the data. The final prediction for an incoming new query is averaged from all the regressors $\hat{f} = \frac{1}{B}\sum_{i=1}^{B} f_B(\mathbf{x})$. Subsampling has the practical effect of decreasing the variance of the model, without increasing its complexity, given that even if the predictions of a single tree are highly sensitive to noise, the average of many trees is not, as long as the trees are not correlated. Bootstrapping achieves this effect by training each tree with a different randomized subsample.

When the individual trees f_b are learned, the building procedure has to create tree nodes that branch the data down the tree; in order to reduce the model variance, only a few features are candidates for splitting at each round. This mitigates the effect that happens when, if just a few features are very strong predictors for y, these features will be selected in many of the B trees, which will become correlated. Given their resilience to noise and outliers, random forest were the best out-of-the-box regressors for the task of predicting cutoff values and query response times, surpassing in effectiveness many other candidates such as kernel ridge regression, Gaussian (regression) processes among others.

We deploy the quantile regression within the same tree framework using gradient boosting regression trees (GBRT). In this case, each tree re-fits the training data using the residuals (gradients) of the training data with respect to the ξ_τ loss function, and a pre-tree weight is calculated using line search. The final decision is a linear combination of the weighted prediction of the tree ensemble.

We used a similar set of features as Culpepper et al. [16]. These features are based on a aggregating statistics for each postings list (such as maximum scores, harmonic/arithmetic mean/median scores, and so on) from a range of similarity functions, along with query specific features such as query length, max score of query terms, and many more. In addition to the TF·IDF, BM25 and query likelihood used in [16], we also build features using Bose-Einstein,

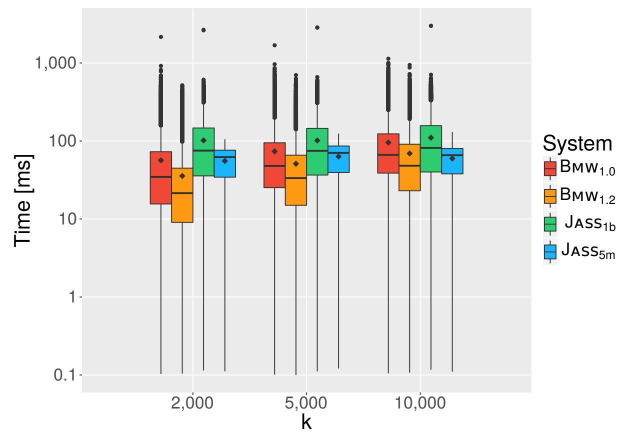

Figure 3: Efficiency comparison of the 31,642 queries from the MQ2009 TREC Task using both aggressive and exact versions of Bmw and Jass. Subscripts denote the aggression parameters (θ for Bmw and ρ for Jass).

DPH, and DFR similarity functions [1]. We also added the geometric mean as an aggregation statistic for each of these similarity functions. We use a total of 147 features in this work.

Tail-Latency Queries in DaaT Traversal Algorithms. This improved approach to predicting k in first stage retrieval is a promising first step to achieving for efficient results without sacrificing effectiveness. However, assuming that the performance of Wand-based algorithms in the first stage is a function of k may not be correct in practice, and other recent work [14] provides persuasive evidence that this assumption is not true in practice. Crane et al. showed that when using Wand and Bmw, tail-latency queries can occur at any k cutoff, making performance guarantees hard to enforce in production systems.

The alternative to using Wand or Bmw in the first stage retrieval is to use a SaaT algorithm such as Jass. Unfortunately, this is not an entirely satisfactory answer either as most of the performance gains in Jass come from using aggressive early termination, which can hurt effectiveness when the number of documents that must be passed into the next stage must also be minimized. So rank safety is yet another confounding factor. DaaT and SaaT processing algorithms can sacrifice effectiveness for efficiency by relaxing the rank-safety constraint. For example, Jass allows a parameter ρ to be set which bounds the maximum number of postings to score per query, and variants of Wand can use a parameter θ (or sometimes F) which induces more aggressive skipping during postings list traversal. So, there is a trade-off between retrieval depth k and rank safety in a pure efficiency sense. This relationship was previously explored by Tonellotto et al. [52], who also used a query difficulty prediction framework to solve the problem. We build on this idea in this work, but also account for the fact that using only Wand based algorithms can still result in poorly performing tail-latency queries. We can see that boosting θ alone does indeed make Bmw faster in Figure 3, but the tail-latency queries remain.

	$\text{Bmw}_{1.1}$	$\text{Bmw}_{1.2}$	$\text{Bmw}_{1.3}$	Jass_{1b}	Jass_{5m}
$\text{Bmw}_{1.0}$	86.0	61.7	46.6	56.2	16.7
$\text{Bmw}_{1.1}$	-	67.4	49.7	53.1	18.3
$\text{Bmw}_{1.2}$	-	-	68.6	42.3	24.2
$\text{Bmw}_{1.3}$	-	-	-	31.4	26.7
Jass_{1b}	-	-	-	-	8.0

Table 1: The percentage overlap of queries that fall in 95th percentile efficiency band for $k = 2{,}000$. Clearly, making Bmw more aggressive may improve timings, but the tail queries are generally similar regardless of the aggression parameter θ. On the other hand, it is less common for Jass and Bmw to have overlapping tail queries, especially when a non-exhaustive ρ value is used.

So, our next task is to explore the likelihood of tail-latency queries when using the MQ2009 topic set. Crane et al. [14] recently did a comparative analysis using the UQV [6] query set and the ClueWeb12B document collection with fixed values of k. We reproduce their work here across our own query set and fixed k values. Figure 3 shows the breakdown of all 31,642 queries across a number of fixed values of k, selected as appropriate sizes for an LtR system [38]. Similar to Crane et al., we observe that the exhaustive Bmw algorithm is superior to the exhaustive Jass algorithm, but the heuristic Jass traversal (with the recommended 10% heuristic) eliminates all tail-latency queries. On the other hand, the aggressive Bmw traversal does improve the mean and median times, but does not reduce the likelihood of tail-latency queries. Note that we selected the value for the heuristic, $\theta = 1.2$, based on other work that shows that more aggressive approaches result in reduced effectiveness [13, 40]. It is also noteworthy that the exhaustive Bmw traversal has a faster median time than the aggressive Jass traversal when $k \leq 5{,}000$.

Additionally, we do a simple overlap analysis on the 95th percentile tail-latency for each algorithm to determine whether each system has similar tail-latency queries. Table 1 shows the percentage of the tail-latency queries that overlap between each system, where $k = 2{,}000$. Exact Jass, exact Bmw and aggressive Bmw tend to share similar tail-latency queries. However, we note that the aggressive Jass traversal tends to share only a small percentage of the tail-latency queries that occur in the other systems. This provides further motivation for our proposed hybrid ISN index configuration.

In light of this new evidence, a pragmatic hypothesis emerges: Can we somehow combine the best properties of Jass and Bmw to create a hybrid approach that captures the best of both worlds?

4 APPROACH

Problem Definition. First, we define the problem. Given a query q, a series of re-ranking stages R, and a target evaluation metric M for the final stage, how can we select both k and the processing algorithm A for the initial (bag-of-words) stage such that k, processing time t, and effectiveness loss L are minimized?

Algorithm 1: Candidate generation pipeline based on predicting k

Input : A query q, a regressor $\mathcal{R}_k$ that predicts the required k for q, a regressor $\mathcal{R}_\rho$ that predicts the required ρ for Jass up to a maximum ρ value ρ_{max}, and a k-threshold T_k

Output: A set of candidate documents, C

$C \leftarrow \emptyset$
$P_k \leftarrow \mathcal{R}_k(q)$
if $P_k > T_k$ **then**
  $P_\rho \leftarrow \mathcal{R}_\rho(q)$
  $C \leftarrow \text{ISN}_{\text{Jass}}(q, P_k, P_\rho)$
else
  $C \leftarrow \text{ISN}_{\text{Bmw}}(q, P_k)$
end
return C

Untangling the objectives. Our first goal is to untangle the objectives, and describe a unified methodology to satisfy all of the constraints in a principled way. We draw inspiration from all of these recent studies. We still want to minimize k as the performance of later stage re-ranking algorithms is sensitive to the number of documents, and we also want to provide performance guarantees on the running time of the first stage ranker. The key observation that pulls these seemingly different objectives together is that the classification approach described by Culpepper et al. actually used classic *query hardness* features for the learning model [10, 40, 52]. Furthermore, the MED approach allows many more queries to be used for training than methods which require a full set of relevance judgments to be available in order to minimize effectiveness loss. So, we explore the possibility of using a single predictive framework to minimize all three constraints in a unified way.

System Architecture. The first major difference in our approach is that we opt to build a hybrid architecture. Work on distributed IR has shown that an effective approach to scaling is to replicate popular ISNs [18, 23, 29, 30]. Here, we assume that we can build ISNs that are optimized for different types of queries. In other words, when we build replicas, we may opt to build a document-ordered index (appropriate for DaaT traversal), or an impact-ordered index (appropriate for SaaT traversal). This idea is key to our novel framework: Selecting algorithm $a \in A$ actually refers to selecting an ISN to process the query which is configured to run algorithm a, and ISN selection is already a common problem in distributed search architectures [7, 28]. In practice, our "Stage-0" predictions would be performed by the resource selection process in a large-scale distributed IR system.

Hybrid Approaches. Based on several observations about the relative performance of Jass and Bmw, we are now in a position to describe a few different hybrid approaches to query processing. Our goal is to limit the disadvantages of each traversal algorithm,

Algorithm 2: Candidate generation pipeline based on predicting both k and run time

Input : A query q, a regressor $\mathcal{R}_k$ that predicts the required k for q, a regressor $\mathcal{R}_\rho$ that predicts the required ρ for JASS up to a maximum ρ value $\rho_{\max}$, a k-threshold T_k, a regressor $\mathcal{R}_t$ that predicts the running time of q, and run-time threshold T_t

Output: A set of candidate documents, C

```
C ← ∅
P_k ← R_k(q)
if k > T_k then
    P_ρ ← R_ρ(q)
    C ← ISN_JASS(q, P_k, P_ρ)
else
    P_t ← R_t(q)
    if P_t > T_t then
        P_ρ ← R_ρ(q)
        C ← ISN_JASS(q, P_k, P_ρ)
    else
        C ← ISN_BMW(q, P_k)
    end
end
return C
```

and exploit the desirable properties. Several different variations were used in our preliminary experiments, and the two best are shown here. In both algorithms, the first step is to predict the k cutoff. If k is greater than the threshold T_k, then proceed to the JASS pipeline as in Algorithm 1, or make a second query difficulty prediction as in Algorithm 2. If JASS is used, a prediction for ρ is made, but capped at $\rho_{\max}$, which allows us to achieve the desired performance guarantees. In our experiments, $\rho_{\max} = 10$ million postings as this requires less than 200ms on our current hardware configuration. The remaining queries are processed using BMW with rank-safety.

5 EXPERIMENTS

We now look at the various predictions that are necessary to achieve our performance requirements. Our performance requirements for effectiveness are to achieve a target MED that is low enough to result in no measurable effectiveness difference for the target metric. Our performance requirements for efficiency are no queries over 200 ms with a 99.99% response time guarantee. That is, we can afford at most 3 over-budget queries for our entire query trace.

Predicting k. First, we validate that our new approach to k prediction using quantile regression is effective. Using our newly devised regression technique, we can compare the efficiency and effectiveness trade-offs between the size of the candidate retrieval set k, and the expected effectiveness loss MED_{RBP}. Figure 4 shows the predictive power of the random forest (RF_ε) and quantile regression (QR_τ) when compared to the oracle results for ϵ target between 0.001 and 0.10, and to using a fixed cutoff for all queries. Note that the graph on the left presents results as the median k result in contrast to the right graph which shows the results for the mean k results as done in previous work. Since the distribution of the true k values is skewed for the queries as shown in Figure 2, presenting the results using the median more accurately captures the trade-offs.

System	RMSE	Precision	Recall	F	M-Precision	M-Recall	M-F	AUC
QR	**0.76**	**0.73**	0.52	**0.62**	**0.87**	**0.76**	**0.81**	**0.98**
RF	0.77	0.71	**0.54**	0.61	0.84	0.76	0.80	0.97
LR	0.84	0.73	0.49	0.58	0.85	0.74	0.79	0.96

Table 2: Regression and tail query classification ($\tau = 0.95$) performance for Quantile Regression, Random Forests and Linear Regression, best values bold (difference may be on the third decimal)

Predicting ρ. Based on the lessons learned when attempting to build a robust predictive framework for k, we now turn our attention to the aggressiveness parameter ρ in JASS. Previous work has shown that using an exhaustive ρ results in effective top-k retrieval, however, using a heuristic ρ can give similar effectiveness, yet much more efficient retrieval [33, 34]. The recommended heuristic value of ρ is 10% of the size of the collection [34], which is around 5 million for the ClueWeb09B collection. Figure 5 shows the distribution of ρ values required to when targeting a $\text{MED-RBP}_{0.95} < 0.001$, or essentially, no measureable difference in the results lists between exhaustive and aggressive JASS traversals. Clearly, the majority of the distribution lies well to the lower side of the 10% heuristic value. This motivates us to predict ρ on a query-by-query basis. Again, we deploy both a Random Forest and a Gradient Quantile Regression method as the distribution of ρ is skewed.

Figure 6 shows the median predicted ρ values compared with the fixed and oracle. Both the QR and RF regression methods manage to improve on the fixed ρ median. Note that when measuring the $\text{MED-RBP}_{0.95}$ for this experiment (and subsequently, training the value of ρ), the k utilized was the optimal value of k from the previous experiment. The reason for using this k is that we must fix k, otherwise our effectiveness scores may change as a result of k, not just ρ. Indeed, this setting of k also allows us to find the true optimal $\text{MED-RBP}_{0.95}$ for JASS, denoted by the oracle point in Figure 6.

Predicting response time. Given that our entire framework is built using query performance prediction features, and we want to minimize tail-latency queries, we explore the accuracy of query performance prediction within the framework.

Table 2 shows the performance of three different regression methods for regressed query times and for predicting whether a query time will fall into the last percentile of the distribution, i.e., if it will be a *tail-latency* or not. We replicate the previous set-up by using exactly the same features as before and 10-fold cross validation. We learn a regressor based on Random Forest, Gradient Quantile Regression, and a Linear Regression, which was employed previously by Macdonald et al. [40] for the same task, although with a smaller set of features.

We report on regression performance using root mean squared error (RMSE) and on a number of binary classification metrics for the tail-latency prediction task. To predict tail-latency queries for the 99th percentile, we learn a threshold in the training set by selecting the minimum running time of all the queries in the 95th percentile. We report on the area under the curve (AUC),

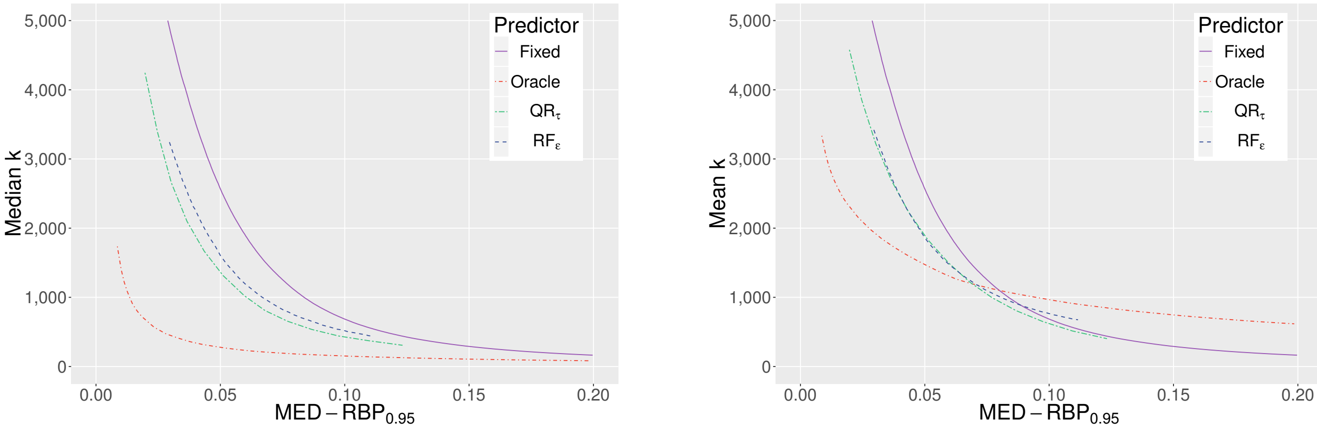

Figure 4: MED_{RBP} versus median k (left) and mean k (right) for all ϵ thresholds between 0.001 and 0.200 when using a Random Forest regression, and for all τ values between 0.10 and 0.75 with $\epsilon = 0.001$ for Quantile Regression, in first stage retrieval for the 31,642 queries from the MQ2009 TREC Task. Note that the Quantile Regression clearly improves the median k (compared with Random Forests) without negatively affecting the mean k.

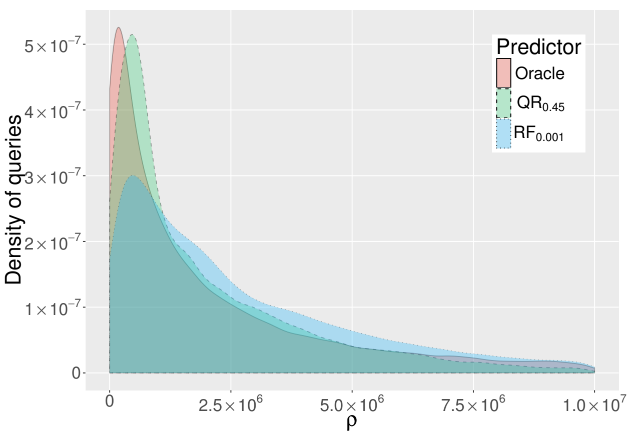

Figure 5: A comparison of the distributions the actual ρ vs the predicted ρ when using a Random Forest regression and a Quantile Regression in first stage retrieval for the 31,642 queries from the MQ2009 TREC Task. Note that the Random Forest uses a training value of $\epsilon = 0.001$, whereas the best-fit distribution for the Quantile Regression was $\tau = 0.45$ for ρ.

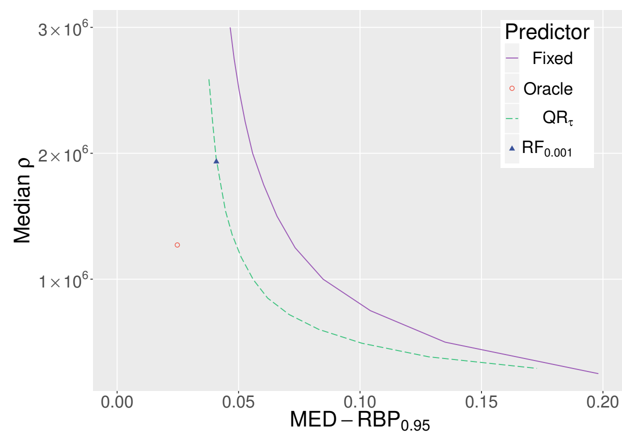

Figure 6: MED_{RBP} versus median ρ for $\epsilon = 0.001$ when using the RF regression, and for all τ values between 0.10 and 0.75 with $\epsilon = 0.001$ for QR, in first stage retrieval for all 31,642 queries. Quantile Regression and Random Forests behave similarly with respect to the median ρ, but QR is still preferred as the final predicted ρ distribution fits better with the idealized results as shown in Figure 5.

precision/recall/F measure for the positive class (the query was a tail-latency query) and class-average (macro) precision/Recall/F-measure.

Results show that our predictors are extremely effective for regressing timings, with random forests and quantile regression having a clear edge over linear regression, both in terms of raw regression error (RMSE) and true positive classification. QR has some advantage over RF given that the distribution of timings is skewed (Figure 3). One discussion point is that we did not attempt to deploy any dynamic features, such as those seen in the DDS prediction framework [28]. We leave this for future work.

Putting it all together. Here, we show that by combining all of our predictions into hybrid first-stage retrieval systems, outlined in Algorithms 1 and 2, we can achieve effectiveness equal to a fixed parameter system, while simultaneously reducing the number of documents that must be passed on to the next stage of the multi-stage retrieval system. Additionally, we show that we can use our framework to mitigate tail-latency queries effectively.

Figure 7 shows the performance for 2 different $MED\text{-}RBP_{0.95}$ cut-offs: 0.05 and 0.10. We also show the performance of the oracle selectors, which all had $MED\text{-}RBP_{0.95}$ scores below 0.02. As before, Jass_{1b}, Jass_{5m} and $\text{Bmw}_{1.0}$ refer to using a fixed k – the k was selected such that the mean MED value was equivalent to the target. We also report the results of the two hybrid systems based on Algorithm 1 (Hybrid_k) and Algorithm 2 (Hybrid_h), which use quantile regression for their predictions. Additionally, Table 3

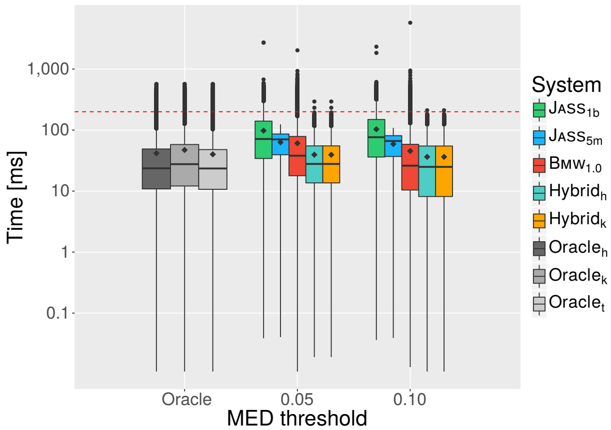

Figure 7: The response time for each system for different bands of MED-RBP$_{0.95}$. Both Hybrid$_k$ and Hybrid$_h$ systems, which predict k and ρ, and k, ρ and time respectively, show a clear improvement over the fixed baselines. Although aggressive JASS has fewer tail latency queries, the k required to attain the given MED values is larger than the exhaustive and hybrid systems, which has implications in the efficiency of the late stage feature extraction and re-ranking. The horizontal line denotes the 200ms budget.

Oracles: MED-RBP$_{0.95}$ < 0.02					
System	**Mean k**	**Median k**	**Mean time**	**Median time**	**% queries > 200 ms**
Oracle$_k$	3334	1735	47.2	27.6	3.1
Oracle$_t$	3334	1735	**40.0**	**23.5**	**2.2**
Oracle$_h$	3334	1735	41.6	23.6	2.6
MED-RBP$_{0.95}$ = 0.05					
System	**Mean k**	**Median k**	**Mean time**	**Median time**	**% queries > 200 ms**
BMW$_{1.0}$	2600	2600	60.9	38.1	4.4
JASS$_{1b}$	2600	2600	97.9	71.4	11.8
JASS$_{5m}$	3100	3100	63.1	70.3	**0**
Hybrid$_k$	**2232**	**1667**	**40.9**	**29.2**	0.006
Hybrid$_h$	**2232**	**1667**	**40.9**	**29.2**	0.006
MED-RBP$_{0.95}$ = 0.10					
System	**Mean k**	**Median k**	**Mean time**	**Median time**	**% queries > 200 ms**
BMW$_{1.0}$	800	800	45.3	26.1	2.4
JASS$_{1b}$	800	800	103.2	74.9	13.2
JASS$_{5m}$	900	900	59.2	65.7	**0**
Hybrid$_k$	**648**	**441**	**36.4**	**24.9**	0.003
Hybrid$_h$	**648**	**441**	**36.4**	**24.9**	0.003

Table 3: Summary statistics for k, time and the % of queries with response times above 200 ms. Each sub-table corresponds to a section of Figure 7, and the best values are bold. Not only do the hybrid systems require less documents in the first stage, they also run more efficiently across the ISNs, and generally reduce tail latencies compared to fixed systems. In particular, the hybrid methods both have only 1 query > 200ms in the MED 0.10 case, and 2 queries > 200ms in the MED 0.05 case.

shows the average and median k, as well as the time characteristics for the systems presented in Figure 7.

Our results show that our hybrid systems both outperform the equivalent fixed BMW or JASS traversals for the given MED targets. For example, with a target of MED-RBP$_{0.95}$ = 0.05, our hybrid systems can achieve a mean and median query response time 20 ms and 8.9 ms below the best fixed system, respectively. The hybrid systems return, on average, 368 less candidate documents to the next stage of the retrieval architecture, resulting in further efficiency gains along the cascade without loss in effectiveness. Finally, our hybrid systems managed to each have only 2 queries that ran longer than our target efficiency of 200 ms, with run times of 232.4 ms and 294.1 ms respectively. Similar outcomes are observed when the MED target is relaxed to 0.10. Although the JASS$_{5m}$ fixed system outperforms our hybrids in reducing tail latencies, it must retrieve a larger number of documents to achieve the same effectiveness target, which has negative implications on the efficiency of the following stages. We note that we do not consider the time required to make our predictions. Recent work using similar models show a prediction overhead of < 0.75 ms per prediction [25]. So, in the worst case, we are likely to only add 2 − 3ms per query.

System	NDCG@10	ERR@10	RBP p = 0.80
`uog-ideal`	0.3578	0.4346	0.4357 (0.1366)
Hybrid$_k$	0.3464	0.4174	0.4231 (0.1523)
Hybrid$_h$	0.3464	0.4174	0.4231 (0.1523)
JASS$_{5m}$	0.3554	0.4354	0.4297 (0.1517)

Table 4: Effectiveness measurements taken across the held-out query set. No statistical significance was measured between the hybrid systems with respect to the ideal system, using the two one-sided test with $p < 0.05$.

Validating Robustness. As a final test of robustness, we run both of our hybrid systems across the 50 (unseen) TREC 2009 Web Track queries. These queries were held out from the train and test procedures reported in earlier sections. Since these queries have judgements to depth 12, we report NDCG@10, ERR@10 and RBP$_{p=0.80}$ [35]. For the hybrid systems, we used the same prediction configuration that was used in the MED-RBP$_{0.95}$ = 0.05 task from Figure 7 and Table 3.

Table 4 shows the effectiveness measurements. Clearly, our hybrid systems have a small loss in effectiveness compared to the ideal end-stage run. In order to test whether the `uog-ideal` run was significantly better than our hybrid runs, we ran the two one-sided test [47] of equivalence (TOST). For each TOST test, we set the ϵ parameter as $\epsilon = 0.1 \cdot \mu$, where μ is the mean effectiveness score of the ideal run for the desired metric. We found that the ideal system was not statistically significantly different than our hybrid systems, with $p < 0.05$.

6 CONCLUSION

We presented and validated a unified framework to predict a wide range of performance-sensitive parameters for early-stage candidate retrieval systems using MED and reference lists as guides for training (RQ1). Preliminary experiments show that the DAAT BMW approach is efficient but suffers from the occasional tail query, which the SAAT JASS algorithm does not. Hybrid systems based on this framework were shown to minimize effectiveness loss while also minimizing query-latency across all stages of a multi-stage search architecture. Given a fixed budget of 200ms for a first-stage response time, we can achieve this budget 99.99% of the time with the hybrid systems, across an index of 50 million documents and a trace of over 30,000 queries, thus answering RQ2 in the affirmative. In particular, we find that using quantile regression (GBRT) for predicting k, ρ and response time allows us to minimize the late stage effectiveness loss while simultaneously minimizing the size of the initial candidate set, thus answering RQ3.

REFERENCES

[1] G. Amati and C. J. Van Rijsbergen. 2002. Probabilistic Models of Information Retrieval Based on Measuring the Divergence from Randomness. 20, 4 (2002), 357–389.

[2] V. N. Anh, O. de Kretser, and A. Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR.* 35–42.

[3] N. Asadi and J. Lin. 2013. Document Vector Representations for Feature Extraction in Multi-Stage Document Ranking. *Inf. Retr.* 16, 6 (2013), 747–768.

[4] N. Asadi and J. Lin. 2013. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. In *Proc. SIGIR.* 997–1000.

[5] N. Asadi, J. Lin, and A. P. De Vries. 2014. Runtime optimizations for tree-based machine learning models. *Trans. on Know. and Data Eng.* 26, 9 (2014), 2281–2292.

[6] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. 2016. UQV100: A Test Collection with Query Variability. In *Proc. SIGIR.* 725–728.

[7] D. Broccolo, C. Macdonald, O. Salvatore, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. 2013. Load-sensitive Selective Pruning for Distributed Search. In *Proc. CIKM.* 379–388.

[8] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM.* 426–434.

[9] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. 2010. Early Exit Optimizations for Additive Machine Learned Ranking Systems.. In *Proc. WSDM.* 411–420.

[10] D. Carmel and E. Yom-Tov. 2010. *Estimating the Query Difficulty for Information Retrieval.* Morgan & Claypool.

[11] K. Chakrabarti, S. Chaudhuri, and V. Ganti. 2011. Interval-based pruning for top-k processing over compressed lists. In *Proc. ICDE.* 709–720.

[12] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proc. CIKM.* 621–630.

[13] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. 2016. Assessing efficiency–effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments. *Inf. Retr.* 19, 4 (2016), 351–377.

[14] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. 2017. A Comparison of Document-at-a-Time and Score-at-a-Time Query Evaluation. In *Proc. WSDM.* To appear.

[15] M. Crane, A. Trotman, and R. O'Keefe. 2013. Maintaining discriminatory power in quantized indexes. In *Proc. CIKM.* 1221–1224.

[16] J. S. Culpepper, C. L. A. Clarke, and J. Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *Proc. ADCS.* 17–24.

[17] V. Dang, M. Bendersky, and W. B. Croft. 2013. Two-stage learning to rank for information retrieval. In *Proc. ECIR.* 423–434.

[18] J. Dean and L. A. Barroso. 2013. The Tail at Scale. *Comm. ACM* 56, 2 (2013), 74–80.

[19] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. 2013. Optimizing Top-k Document Retrieval Strategies for Block-Max Indexes. In *Proc. WSDM.* 113–122.

[20] S. Ding, J. Attenberg, and T. Suel. 2010. Scalable Techniques for Document Identifier Assignment in Inverted Indexes. In *Proc. WWW.* 311–320.

[21] S. Ding and T. Suel. 2016. Faster Top-k Document Retrieval Using Block-Max Indexes. In *Proc. SIGIR.* 993–1002.

[22] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. 2011. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proc. VLDB* 4, 12 (2011), 1213–1224.

[23] G. Francès, X. Bai, B. B. Cambazoglu, and R. Baeza-Yates. 2014. Improving the Efficiency of Multi-site Web Search Engines. In *Proc. WSDM.* 3–12.

[24] S.-W. Hwang, K. Saehoon, Y. He, S. Elnikety, and S. Choi. 2016. Prediction and Predictability for Search Query Acceleration. *ACM Trans. Web* 10, 3 (Aug. 2016), 19:1–19:28.

[25] M. Jeon, S. Kim, S.-W. Hwang, Y. He, S. Elnikety, A.L. Cox, and S. Rixner. 2014. Predictive Parallelization: Taming Tail Latencies in Web Search. In *Proc. SIGIR.* 253–262.

[26] X. Jin, T. Yang, and X. Tang. 2016. A Comparison of Cache Blocking Methods for Fast Execution of Ensemble-based Score Computation. In *Proc. SIGIR.* 629–638.

[27] A. Kane and F. Wm. Tompa. 2014. Distribution by Document Size. In *LSDS-IR.*

[28] S. Kim, Y. He, S.-W. Hwang, S. Elnikety, and S. Choi. 2015. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In *Proc. WSDM.* 7–16.

[29] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Efficient distributed selective search. *Inf. Retr.* (2016), 1–32.

[30] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Load-Balancing in Distributed Selective Search. In *Proc. SIGIR.* 905–908.

[31] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. 2013. Online controlled experiments at large scale. In *Proc. KDD.* 1168–1176.

[32] D. Lemire and L. Boytsov. 2015. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.* 45, 1 (2015), 1–29.

[33] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. (2016).

[34] J. Lin and A. Trotman. 2015. Anytime Ranking for Impact-Ordered Indexes. In *Proc. ICTIR.* 301–304.

[35] X. Lu, A. Moffat, and J. S. Culpepper. 2016. The Effect of Pooling and Evaluation Depth on IR Metrics. *Inf. Retr.* 19, 4 (2016), 416–445.

[36] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2015. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. In *Proc. SIGIR.* 73–82.

[37] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2016. Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles. In *Proc. SIGIR.* 833–836.

[38] C. Macdonald, R. L. T. Santos, and I. Ounis. 2013. The whens and hows of learning to rank for web search. *Inf. Retr.* 16, 5 (2013), 584–628.

[39] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. 2013. About learning models with multiple query-dependent features. *ACM Trans. Information Systems* 31, 3 (2013), 11.

[40] C. Macdonald, N. Tonellotto, and I. Ounis. 2012. Learning to Predict Response Times for Online Query Scheduling. In *Proc. SIGIR.* 621–630.

[41] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. 2015. Efficient location-aware web search. In *Proc. ADCS.* 4.1–4.8.

[42] A. Moffat and J. Zobel. 2008. Rank-Biased Precision for Measurement of Retrieval Effectiveness. *ACM Trans. Information Systems* 27, 1 (2008), 2.1–2.27.

[43] J. Pedersen. 2010. Query understanding at Bing. *Invited talk, SIGIR* (2010).

[44] M. Petri, J. S. Culpepper, and A. Moffat. 2013. Exploring the Magic of WAND. In *Proc. ADCS.* 58–65.

[45] M. Petri, A. Moffat, and J. S. Culpepper. 2014. Score-safe term dependency processing with hybrid indexes. In *Proc. SIGIR.* 899–902.

[46] C. Rossi, E.S. de Moura, A.L. Carvalho, and A.S. da Silva. 2013. Fast Document-at-a-time Query Processing Using Two-tier Indexes. In *Proc. SIGIR.* 183–192.

[47] D. J. Schuirmann. 1987. A comparison of the Two One-Sided Tests Procedure and the Power Approach for assessing the equivalence of average bioavailability. *J. Pharmacokinetics and Biopharmaceutics* 15, 6 (1987), 657–680.

[48] A. Shtok, O. Kurland, and D. Carmel. 2016. Query Performance Prediction Using Reference Lists. *ACM Trans. Information Systems* 34, 4 (2016), 19.1–19.34.

[49] F. Silvestri. 2007. Sorting Out the Document Identifier Assignment Problem. In *Proc. ECIR.* 101–112.

[50] L. Tan and C. L. A. Clarke. 2015. A Family of Rank Similarity Measures Based on Maximized Effectiveness Difference. *Trans. on Know. and Data Eng.* 27, 11 (2015), 2865–2877.

[51] N. Tonellotto, C. Macdonald, and I. Ounis. 2011. Effect of Different Docid Orderings on Dynamic Pruning Retrieval Strategies. In *Proc. SIGIR.* 1179–1180.

[52] N. Tonellotto, C. Macdonald, and I. Ounis. 2013. Efficient and effective retrieval using selective pruning. In *Proc. WSDM.* 63–72.

[53] A. Trotman. 2014. Compression, SIMD, and Postings Lists. In *Proc. ADCS.* 50:50–50:57.

[54] A. Trotman, X.-F. Jia, and M. Crane. 2012. Towards an efficient and effective search engine. In *Wkshp. Open Source IR.* 40–47.

[55] A. Trotman and J. Lin. 2016. In Vacuo and In Situ Evaluation of SIMD Codecs. In *Proc. ADCS.* 1–8.

[56] L. Wang, J. Lin, and D. Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proc. SIGIR.* 105–114.

[57] Q. Wang, C. Dimpoloulos, and T. Suel. 2016. Fast First-Phase Candidate Generation for Cascading Rankers. In *Proc. SIGIR.* 295–304.

[58] W. Webber, A. Moffat, and J. Zobel. 2010. A Similarity Measure for Indefinite Rankings. *ACM Trans. Information Systems* 28, 4 (Nov. 2010), 20.1–20.38.

[59] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. 2014. Classifier cascades and trees for minimizing feature evaluation cost. *J. of Machine Learning Research* 15 (2014), 2113–2144.

[60] J.-M. Yun, Y. He, S. Elnikety, and S. Ren. 2015. Optimal Aggregation Policy for Reducing Tail Latency of Web Search. In *Proc. SIGIR*. 63–72.

[61] J. Zhang, X. Long, and T. Suel. 2008. Performance of Compressed Inverted List Caching in Search Engines. In *Proc. WWW*. 387–396.

[62] J. Zobel and A. Moffat. 2006. Inverted Files for Text Search Engines. *ACM Comp. Surv.* 38, 2 (2006), 6.1–6.56.