Ranked List Truncation for Large Language Model-based Re-Ranking

Chuan Meng
University of Amsterdam
Amsterdam, The Netherlands
c.meng@uva.nl

Negar Arabzadeh
University of Waterloo
Waterloo, Canada
narabzad@uwaterloo.ca

Arian Askari
Leiden University
Leiden, The Netherlands
a.askari@liacs.leidenuniv.nl

Mohammad Aliannejadi
University of Amsterdam
Amsterdam, The Netherlands
m.aliannejadi@uva.nl

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

ABSTRACT

In this paper, we explore ranked list truncation (RLT) in a novel "*retrieve-then-re-rank*" perspective, where we optimize re-ranking by truncating the retrieved list (i.e., trim re-ranking candidates). RLT is crucial for re-ranking as it can improve re-ranking efficiency by sending variable-length candidate lists to a re-ranker on a per-query basis, and has the potential to improve re-ranking effectiveness. Despite its importance, there is limited research into applying RLT methods to this new perspective. To address this research gap, we reproduce existing RLT methods in the context of re-ranking, especially newly emerged large language model (LLM)-based re-ranking. In particular, we examine to what extent established findings on RLT for retrieval are generalizable to the "retrieve-then-re-rank" setup from three perspectives: (i) assessing RLT methods in the context of LLM-based re-ranking with lexical first-stage retrieval, (ii) investigating the impact of different types of first-stage retrievers on RLT methods, and (iii) investigating the impact of different types of re-rankers on RLT methods. We perform experiments on the TREC 2019 and 2020 deep learning tracks, investigating 8 RLT methods for pipelines involving 3 retrievers and 2 re-rankers. We reach new insights into RLT methods in the context of re-ranking.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**; **Language models**; **Retrieval efficiency**; **Retrieval effectiveness**.

KEYWORDS

Ranked list truncation, Re-ranking, Large language models

ACM Reference Format:
Chuan Meng, Negar Arabzadeh, Arian Askari, Mohammad Aliannejadi, and Maarten de Rijke. 2024. Ranked List Truncation for Large Language Model-based Re-Ranking. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24), July 14–18, 2024, Washington, DC, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3626772.3657864

1 INTRODUCTION

Ranked list truncation (RLT), a.k.a. query cut-off prediction [10, 23], has been studied for over two decades [3, 32] and recently attracted lots of attention in the information retrieval (IR) community [6, 7, 25, 29, 57, 60]. The task of RLT is to determine how many items in a ranked list should be returned such that a user-defined metric is optimized [7]. The user-defined metric typically considers the balance between the utility of search results and the cost of processing search results [57]. RLT is crucial in various IR applications where it is money- or time-consuming to review a returned item [60]. E.g., in patent search [26] or legal search [54, 57], providing a ranked list with an overwhelming number of items is too costly for patent experts or litigation support professionals [3].

A new angle. Existing studies mainly focus on RLT for single-stage retrieval, i.e., optimizing a user-defined metric (e.g., F1) of a retrieved list by truncating it at a certain position. In this paper, we focus on RLT for *re-ranking* [5], i.e., RLT in a "*retrieve-then-re-rank*" setup, as shown in Figure 1. In this setup, we still truncate a given retrieved list but focus on enhancing trade-offs between effectiveness and efficiency in re-ranking; truncating the retrieved list directly translates to a reduction in re-ranking depth. RLT for re-ranking is important because: (i) RLT can improve re-ranking efficiency by sending variable-length lists of candidates to a re-ranker on a per-query basis; re-rankers are typically computationally expensive [25] and particularly, recently proposed large language model (LLM)-based re-rankers [27, 46–48, 53, 63, 65] with billions of parameters lead to a substantial increase in computational overhead [67], making it hard to apply them in practice; applying a fixed re-ranking cut-off to all queries is a common practice in the literature; however, individual queries can be answered effectively by a shorter or a longer list of re-ranking candidates [9], so RLT can avoid unnecessary re-ranking costs by dynamically trimming the retrieved list; and (ii) RLT has the potential to improve re-ranking effectiveness; indeed, feeding a long retrieved list that includes many irrelevant items to a re-ranker instead can result in inferior re-ranking quality than a shorter retrieved list [62] .

Despite its importance, limited research has explored the application of RLT methods in the "*retrieve-then-re-rank*" setup [13, 24, 59, 62]. E.g., Zamani et al. [62] only use one RLT method to truncate retrieved lists from BM25 to improve the performance of BERT-based re-ranking [42]. Put differently, there is a lack of systematic and comprehensive studies into the use of RLT methods that have

SIGIR '24, July 14–18, 2024, Washington, DC, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0431-4/24/07.
https://doi.org/10.1145/3626772.3657864

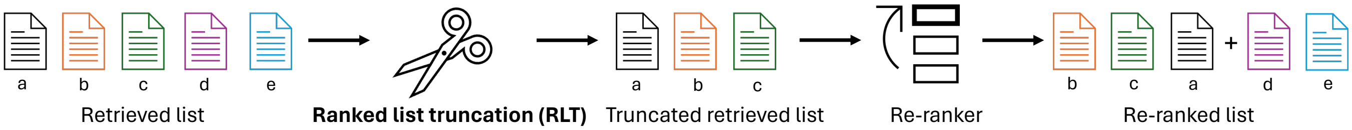


Figure 1: A schematic diagram of RLT in the "*retrieve-then-re-rank*" setup.

originally been introduced to optimize retrieval, in the context of re-ranking, especially newly emerged LLMs-based re-ranking.

Research goal. In this reproducibility paper, we examine *to what extent established findings on RLT for retrieval are generalizable to the "retrieve-then-re-rank" setup*. Specifically, we study the following **findings** from the literature on RLT: (i) Supervised RLT methods generally perform better than their unsupervised counterparts (e.g., set a fixed cut-off for all queries) [6, 25, 57, 60]. (ii) Distribution-based supervised RLT methods (i.e., directly predict a distribution among all candidate cut-off points) perform better than their sequential labeling-based counterpart (i.e., predict whether to truncate at each candidate point) [6, 57, 60]. (iii) Jointly learning RLT with other tasks (e.g., predicting the relevance of each item in the retrieved list) results in better RLT quality [57]. (iv) When truncating a retrieved list returned by a neural-based retriever, incorporating its embeddings improves RLT quality [29].

Reproducibility challenge. We highlight the main challenges of applying RLT methods from optimizing retrieval to optimizing re-ranking: (i) the new "retrieve-then-re-rank" setup leads to a new optimization goal for RLT methods, i.e., improving the trade-offs between effectiveness and efficiency in the re-ranking process; more importantly, a specific trade-off can be considered as the optimization goal to meet the requirements of a specific scenario, e.g., effectiveness is more important than efficiency in professional search than web-search; and (ii) the re-ranking setup introduces the *type* of re-ranker as a factor that influences RLT quality; also, it is important to investigate the impact of the interaction between retrievers and re-rankers on RLT; thus, it is important to explore RLT performance under different pipelines of widely-used retrievers, e.g., lexical, leaned sparse [17] and dense [27] retrievers, and different re-rankers, e.g., LLM-based [27] or pre-trained language model-based re-rankers [43].

Scope. We consider the challenges and examine each established finding from the literature on RLT in three settings: (i) we begin by checking if RLT methods optimizing for different trade-offs between effectiveness and efficiency of a state-of-the-art LLM-based re-ranker, RankLLaMA [27], with a lexical first-stage retriever; next, to study the impact of retriever types on RLT methods, we assess RLT methods for the LLM re-ranker with other types of retrievers, i.e., learned sparse (SPLADE++ [17]), and dense (RepLLaMA [27]) retrievers; and finally, to study the impact of the choice of re-rankers on RLT methods, we assess RLT methods for a widely-used pre–trained language model-based re-ranker, monoT5 [43]. We perform all experiments on the TREC 2019 and 2020 deep learning (TREC-DL) tracks [11, 12] and consider 8 RLT methods and pipelines involving 3 retrievers and 2 re-rankers, leading to various configurations.

Lessons. Our experiments reveal that findings on RLT do not generalize well to the "*retrieve-then-re-rank*" setup. E.g., we found supervised RLT methods do not show a clear advantage over using a fixed re-ranking depth; potential fixed re-ranking depths are able to closely approximate the effectiveness/efficiency trade-offs achieved by supervised RLT methods. Moreover, we found the choice of retriever has a substantial impact on RLT for re-ranking: with an effective retriever like SPLADE++ or RepLLaMA, a fixed re-ranking depth of 20 can already yield an excellent effectiveness/efficiency trade-off; increasing the fixed depth do not significantly improve effectiveness. An error analysis reveals that supervised RLT methods tend to fail to predict when not to carry out re-ranking; moreover, they seem to suffer from a lack of training data.

Contributions. Our main contributions are as follows:

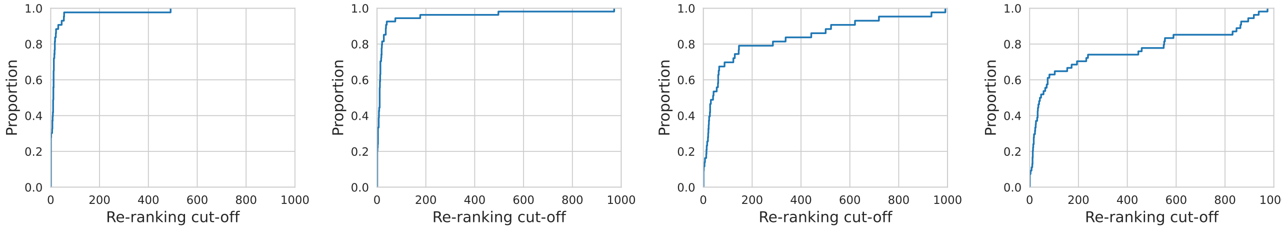
- We reproduce a comprehensive set of RLT methods in a *"retrieve-then-re-rank"* perspective.
- We conduct an empirical analysis with a state-of-the-art LLM-based re-ranker, revealing that setting fixed re-ranking cut-offs results in unnecessary computational costs and diminishes re-ranking quality.
- We conduct extensive experiments on 2 datasets, 8 RLT methods and pipelines involving 3 retrievers and 2 re-rankers, allowing a comprehensive understanding of how RLT methods generalize to the new perspective. We open source our code and data at https://github.com/ChuanMeng/RLT4Reranking.

2 MOTIVATION

In the literature, applying a fixed re-ranking cut-off to all queries is a common practice [27, 28, 46–48, 53, 63, 65]; however, individual queries may need a shorter or a longer list of re-ranking candidates [9]. We conduct an empirical analysis to demonstrate how RLT holds the potential to enhance both the effectiveness and efficiency in re-ranking compared to fixed cut-offs. To do so, we analyze two "retrieve-then-re-rank" pipelines on the TREC-DL 19 and 20 datasets. We use an LLM-based re-ranker (RankLLaMA [27]) in both pipelines, but for first-stage retrieval, we employ a lexical retriever (BM25) in one pipeline and an LLM-based dense retriever (RepLLaMA [27]) in the other.

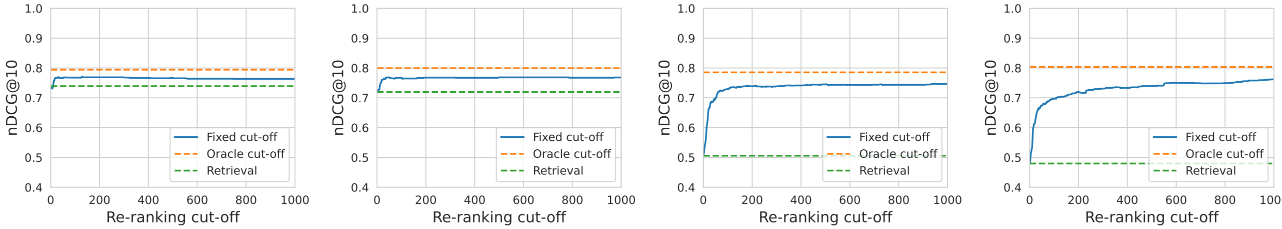
Query-specific cut-offs improve efficiency. We study an *oracle* setup in which we define the oracle as the minimum re-ranking cut-offs yielding the highest nDCG@10 values. we find that individual queries have different oracle cut-offs with a wide range. Thus, a fixed cut-off either wastes computational resources or compromises re-ranking quality for queries that need a deeper cut-off. Figure 2 illustrates the cumulative distribution of oracle cut-offs for both pipelines on both datasets. Interestingly, about 30% of queries do not need re-ranking with RepLLaMA as the retriever, and approximately 5% with BM25; thus, calling expensive re-rankers can be omitted for these queries.

Query-specific cut-offs improve effectiveness. Figure 3 illustrates the comparison of re-ranking quality between using oracle and fixed cut-offs. We find that oracle cut-offs always perform *statistically significantly* (paired t-test, $p < 0.05$) better than all fixed cut-offs in terms of nDCG@10. Hence, a deeper re-ranking cut-off



(a) TREC-DL 19 **(b) TREC-DL 20** **(c) TREC-DL 19** **(d) TREC-DL 20**

Figure 2: Cumulative distribution function of oracle cut-offs for RepLLaMA–RankLLaMA (a, b) and BM25–RankLLaMA (c, d) on TREC-DL 19 and 20. The oracle cut-offs are the minimum re-ranking cut-offs that yield the highest nDCG@10 values.



(a) TREC-DL 19 **(b) TREC-DL 20** **(c) TREC-DL 19** **(d) TREC-DL 20**

Figure 3: nDCG@10 values for RepLLaMA–RankLLaMA (a, b) and BM25–RankLLaMA (c, d) w.r.t. re-ranking cut-offs on TREC-DL 19 and 20.

does not consistently result in improvement and can even be detrimental to re-ranking quality. Our finding is consistent with Zamani et al. [62]. While one might argue that the re-ranking results with a deeper cut-off might be underestimated because of the limited number of judged items within the top 10 ranks, i.e., judged@10 [47], we find that RankLLaMA's judged@10 values for using a fixed cut-off at 1000 and oracle cut-offs are similar, e.g., 95.35% vs.96.05% and 97.41% vs. 97.41% when RepLLaMA as the retriever on TREC-DL 19 and 20, respectively.

RLT methods truncate the retrieved list (i.e., trim re-ranking candidates) on a per-query basis, suggesting that effective RLT has the potential to improve re-ranking efficiency and effectiveness.

3 PRELIMINARIES AND TASK DEFINITION

We recap the task definition of RLT and demonstrate the transition from optimizing retrieval to optimizing re-ranking.

RLT for retrieval. Given a user query q, a collection C containing $|C|$ items, and a retrieved list $L = [d_1, d_2, \ldots, d_{|L|}]$ with $|L|$ ($|L| \ll |C|$) items induced by a first-stage retriever over C in response to q. An RLT approach f aims to predict a truncation point k that maximizes a target metric that is about the retrieved list L itself [6, 25, 29, 57, 60], formally:

$$k = f([x_1, x_2, \ldots, x_{|L|}]) \in \{1, 2, \ldots, |L|\}, \tag{1}$$

where $[x_1, x_2, \ldots, x_{|L|}]$ are item features corresponding one-to-one with the items in the retrieved list $L = [d_1, d_2, \ldots, d_{|L|}]$. Typically, x includes the retrieval score [6] and item statistics [25, 57, 60]. As for the target metric, $F1@k$ and $DCG@k$ have been widely used in prior studies [6, 7, 25, 29, 57, 60]. E.g., $F1@k$ is calculated as:

$$\begin{aligned} F1@k =& \frac{2 \cdot P@k \cdot R@k}{P@k + R@k}, \\ P@k =& \frac{1}{k}\sum_{i=1}^{k} \mathbb{I}(y_i = 1), R@K = \frac{1}{N_L}\sum_{i=1}^{k} \mathbb{I}(y_i = 1), \end{aligned} \tag{2}$$

where $y_i \in \{0, 1\}$ is the relevance label for item d_i in the truncated retrieved list, and N_L denotes the number of relevant items in the retrieved list L.

RLT for re-ranking. In the "*retrieve-then-re-rank*" setup, we no longer focus on optimizing the retrieved list L, but we aim to test the capability of the RLT in optimizing the trade-offs between effectiveness and efficiency in re-ranking. As shown in Figure 1, the truncated retrieved list $L_{1:k} = [d_1, \ldots, d_k]$, serving as re-ranking candidates, is further forwarded to a re-ranker that returns a re-ranked list $\hat{L}_{1:k}$. We append the partial list $L_{k+1:|L|}$ from the retrieved list L that is not re-ranked to $\hat{L}_{1:k}$.

The target metric should evaluate the ranking quality of the re-ranked list $\hat{L}_{1:k}$ (with $L_{k+1:|L|}$) in terms of an IR evaluation metric (e.g., nDCG@10), and measure the computational cost of re-ranking.

4 REPRODUCIBILITY METHODOLOGY

We state our research questions, the experiments designed to address them, and our experimental setup.

4.1 Research questions and experimental design

RQ1 Do RLT methods generalize to the context of LLM-based re-ranking with a *lexical first-stage retriever* when optimized for different effectiveness/efficiency trade-offs?

To address **RQ1**, we first quantify the trade-off between re-ranking effectiveness and efficiency, and then optimize RLT methods to model different trade-offs between effectiveness and efficiency, simulating different requirements and scenarios; then, we evaluate their predicted truncation positions in terms of effectiveness and efficiency in LLM-based re-ranking with a lexical retriever.

RQ2 Do RLT methods generalize to the context of LLM-based re-ranking with *learned sparse or dense first-stage* retrievers when optimized for the different trade-offs, and how does it compare to the case of a lexical retriever?

For answering **RQ2**, we still optimize RLT methods for different trade-offs of the LLM-based re-ranker used in **RQ1**, but study their performance given learned sparse or dense retrievers, and compare the results with those of using a lexical retriever.

RQ3 Do RLT methods generalize to the context of *pre-trained language model-based re-ranking*, and how does it compare to the case of an LLM-based re-ranker?

We address **RQ3** by evaluating the truncation points predicted by RLT methods w.r.t. effectiveness and efficiency in the context of a widely-used pre-trained language model-based re-ranker, and compare the results with those of the LLM-based re-ranker.

4.2 Experimental setup

RLT approaches. We reproduce a variety of unsupervised and supervised RLT methods [6, 7, 25, 29, 57, 60].

We consider the following unsupervised RLT methods.

- **Fixed-*k*** [25] applies a fixed re-ranking cut-off to all queries; we follow common practice and consider cut-offs that are widely used in the literature about re-ranking, namely 10 [28], 20 [28, 46, 47], 100 [16, 27, 46–48, 53, 63, 65–67], 200 [27], 1000 [50].
- **Greedy-*k*** [25] uses the fixed truncation position k that maximizes the target metric value on a training set.
- **Surprise** [7] first calibrates retrieval scores by using generalized Pareto distributions in extreme value theory [45], and truncates a ranked list using a score threshold.

We consider the following supervised RLT methods:

- **BiCut** [25] is a *sequential labeling-based* method; it uses a bidirectional long short-term memory (LSTM) to encode item features over a ranked list, and optimizes the LSTM make a binary prediction (continue or truncate) at each position in a ranked list.
- **Choppy** [6] is a *distribution-based* method, which directly predicts the distribution among all candidate cut-off points, using a transformer encoder [56] to encode item features over a ranked list and predicts the distribution.
- **AttnCut** [60] is also *distribution-based*, encoding item features using a bidirectional LSTM and a transformer encoder.
- **MtCut** [57] is also *distribution-based* and similar to AttnCut, but jointly trains the RLT task with two auxiliary tasks: predicting the relevance of each item in the ranked list and increasing the margin between relevant and irrelevant items. We use the MMoECut variant due to its superior performance.
- **LeCut** [29] is another *distribution-based* and similar to AttnCut, but can only work with a neural-based retriever and incorporates its query–item embeddings as one of the item features. Ma et al. [29] further optimize LeCut with an learning-to-rank (LtR) model jointly. We omit this phase for a fair comparison since other methods are trained without signals from an external LtR model.

We also include **Oracle**, which truncates the retrieved list at the earliest position maximizing re-ranking quality per query.

Optimizing effectiveness/efficiency trade-offs. The leading challenge of adapting RLT methods in the context of re-ranking is to optimize RLT methods with a specific trade-off between effectiveness and efficiency. To solve this challenge, we need to score each truncation point (i.e., re-ranking candidate cut-off) under different effectiveness/efficiency trade-offs. To do so, we quantify different trade-offs using the efficiency-effectiveness trade-off (EET) metric [58] and then compute EET scores at a specific trade-off for all re-ranking candidate cut-offs.

EET is defined for as the weighted harmonic mean of effectiveness σ and efficiency γ measures:

$$EET = \frac{(1+\beta^2)\cdot(\gamma\cdot\sigma)}{\beta^2\cdot\sigma+\gamma}, \tag{3}$$

where β is a hyperparameter to control the relative importance of effectiveness and efficiency, where $\beta = 0$ only considers effectiveness and as it increases more attention is paid to to efficiency. EET requires instantiation of σ and γ based on the specific use case [58]. We follow [58] to instantiate efficiency γ using "exponential decay":

$$\gamma = \exp(\alpha \cdot k), \tag{4}$$

where $k \in \{1, 2, \ldots, |L|\}$ is a truncation point (i.e., re-ranking candidate cut-off) in the given retrieved list L, and $\alpha < 0$ is a hyperparameter to control how rapidly the efficiency measure decreases; we set α to -0.001. We instantiate effectiveness σ as the re-ranking gain with a cut-off k, which is quantified by the difference of re-ranking performance with a cut-off k minus the performance without re-ranking; the performance is in terms of an IR evaluation metric (e.g., nDCG@10).

Therefore, we can adjust β in Equation 3 and α in Equation 4 in EET to quantify different effectiveness/efficiency trade-offs in re-ranking, so as to generate EET value distributions across all cut-off candidates under different trade-offs. As illustrated in Figure 4, we consider three trade-offs between effectiveness and efficiency: β=0 (emphasizing effectiveness), 1 (weighting effectiveness and efficiency equally), and 2 (prioritizing efficiency). With the help of EET value distributions under the three trade-offs, we optimize all distribution-based RLT methods (Choppy [6], AttnCut [60], MtCut [57], LeCut [29]) and Greedy-k for the three trade-offs.

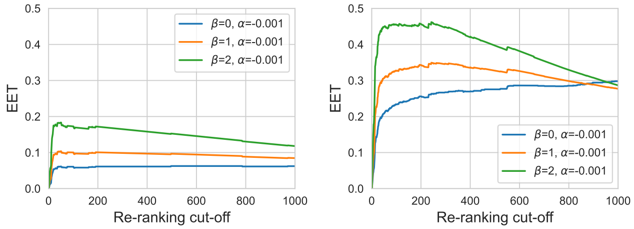
However, the sequential labeling-based RLT method BiCut [25] cannot optimize a EET value distribution. During training, BiCut optimizes the probability of "continue" and "truncation" at each position in a ranked list via the following loss:

$$\mathcal{L} = \sum_{i=1}^{|L|}(\eta\mathbb{I}(y_i = 0)\frac{p_i}{1-r} + (1-\eta)\frac{1-p_i}{r}\mathbb{I}(y_i = 1)), \tag{5}$$

where $y_i \in \{0, 1\}$ is the relevance label for an item at a position it, h p_i is the "continue" probability at a position i, and r is the proportion of relevant items in the ranked list; $\eta \in [0, 1]$ is a hyperparameter to control the balance between "continue" and "truncation". We optimize BiCut for different effectiveness/efficiency trade-offs by adjusting η values, e.g., BiCut trained with a high η value tends to truncate a retrieved list earlier, resulting in more efficiency. Specifically, we consider η=0.4 (emphasizing effectiveness), 0.5 (balancing effectiveness and efficiency), and 0.6 (prioritizing efficiency).[1]

Datasets. We experiment with 2 widely-used IR datasets, TREC 2019 and 2020 deep learning (TREC-DL) tracks [11, 12].[2] These datasets offer relevance judgments in multi-graded relevance scales

[1] We also explore η values of 0.3 and 0.7. BiCut trained with the former tends not to truncate the retrieved list at all, while BiCut trained with the latter tends to truncate the entire retrieved list. [2] We also conducted experiments on Robust04 and draw a similar conclusion as TREC-DL 19 and 20; due to space constraints, we show the result on Robust04 in our repository.



(a) RepLLaMA–RankLLaMA **(b) BM25–RankLLaMA**

Figure 4: Average EET values across TREC-DL 20 queries w.r.t. re-ranking cut-offs. We use nDCG@10 in effectiveness σ. β values 0, 1 and 2 represent prioritizing effectiveness, balancing effectiveness and efficiency, and emphasizing efficiency, respectively.

per query. TREC-DL 19 and 20 are built upon MS MARCO V1 passage ranking collection encompassing 8.8 million passages.

Choice of retrievers. Regarding retrievers, we employ three distinct types: a lexical-based retriever BM25 [49], a learned sparse retriever SPLADE++ ("EnsembleDistil") [17] and an LLM-based dense retriever RepLLaMA (7B) [27]. To increase the comparability and reproducibility of our paper, we obtain retrieval results of BM25 and SPLADE++ using the publicly available resource from Pyserini[3] and get retrieval results of RepLLaMA from Tevatron;[4] each retriever returns 1000 items per query.

Choice of re-rankers. For **RQ1**, **RQ2**, we employ a state-of-the-art LLM-based point-wise reranker, RankLLaMA (7B) [27] and use the resource from Tevatron. For **RQ3**, we employ a widely-used pre-trained language model-based re-ranker, monoT5 ("monot5-base-msmarco") [43] and use the resource from PyGaggle.[5]

Evaluation metrics. We measure re-ranking effectiveness using nDCG@10, the official evaluation metric in TREC deep learning tracks [11, 12], and a widely employed metric in ranking literature [27, 43, 47]. We follow [67] to evaluate re-ranking efficiency by calculating the average re-ranking cut-off across all test set queries, i.e., the number of the average number of re-ranking inferences per query. This consideration is driven by the fact that the re-rankers we employ in this paper are point-wise, and the time spent in point-wise re-ranking is directly proportional to the length of a re-ranking cut-off (i.e., the length of a truncated retrieved list) [31]. We additionally gauge the efficiency by measuring per-query latency; all latency measurements exclude the time to load data and models. We do not consider the latency of first-stage retrieval. Note that RLT methods are lightweight with significantly fewer parameters compared to state-of-the-art re-rankers; the latency introduced by RLT methods can be neglected in the "retrieve-then-re-rank" setup.

Implementation details. We pass the top-1000 retrieved items to all RLT methods per query because 1000 is typically the deepest re-ranking depth in the literature [27, 44, 50]. Note that Suprise [7] only depends on retrieval scores and uses a score threshold for truncation; the score threshold, set based on Cramer-von-Mises statistic testings [14], is not a tunable hyperparameter; thus, Suprise cannot be tuned for different effectiveness/efficiency trade-offs.

For all supervised RLT methods, we use identical item features to eliminate confounding factors from the input; each item is represented by its retrieval score, length, unique token count, and the cosine similarity between its tf-idf/doc2vec [22] vector and the vectors of its adjacent items. We follow [57, 60] to use gensim[6] for computing tf-idf and doc2vec vectors for each item; The dimension of the tf-idf vectors on the MS MARCO V1 collection is 846,221; we follow [57] to set the dimension of doc2vec vectors as 128. LeCut relies on query-item embeddings from a neural retriever as extra item features; thus, we provide LeCut with the concatenation of query and item embeddings from RepLLaMA. Note that we follow all original work to set hyperparameters: for BiCut, we set # Bi-LSTM layers to 2 and the LSTM hidden size to 128; we train BiCut via Adam [21] with a learning rate of 1×10^{-4}; for Choppy, we set # transformer layers to 3, # transformer heads to 8, and transformer hidden size to 128; we train Choppy via Adam with a learning rate of 1×10^{-3}; for AttnCut, we set # Bi-LSTM layers to 2, the LSTM hidden size to 128, # transformer heads to 4, and the transformer hidden size to 128; MtCut and LeCut share almost the same hyperparameters with AttnCut; we train AttnCut, MtCut and LeCut via Adam with a learning rate of 3×10^{-5}. We train all supervised RLT methods for 100 epochs using a batch size of 64 on TREC-DL 19, then infer them on TREC-DL 20, and vice versa. All methods are trained/inferred on an NVIDIA A100 GPU (40GB).

5 RESULTS AND DISCUSSIONS

5.1 RLT for LLM-based re-ranking

To answer **RQ1**, we evaluate RLT methods (optimized for the three effectiveness/efficiency trade-offs) in the context of an LLM-based re-ranker (RankLLaMA [27]) with a lexical retriever (BM25). We report the results on TREC-DL 19 and 20 in Table 1; we also plot the result on TREC-DL 20 in Figure 5. We have three observations.

First, compared to unsupervised RLT methods, supervised RLT one only shows an advantage at achieving better re-ranking effectiveness at a less re-ranking cost in the scenario emphasizing effectiveness; nevertheless, alternative fixed re-ranking depths can deliver results on par with those obtained through supervised methods. (i) In the scenario where effectiveness is prioritized (β=0 and η=0.4), supervised RLT methods achieve better re-ranking effectiveness while maintaining less re-ranking cost. For instance, Choppy (β=0) and AttnCut (β=0) show no significant difference from Fixed-*k* (1000) in terms of nDCG@10 on TREC-DL 20, but with only 69% and 86% of the re-ranking cost of Fixed-*k* (1000), respectively. (ii) In the other scenarios where efficiency received more attention (β=1/2 and η=0.5/0.6), supervised methods do not show an obvious advantage than unsupervised counterparts. E.g., while AttnCut (β=2) and MtCut (β=2) manage to achieve nDCG@10 values comparable to Fixed-*k* (100) using roughly half the re-ranking cost on TREC-DL 20, Greedy-*k* (β=2) attains very similar results as AttnCut and MtCut. (iii) Moreover, as illustrated in Figure 5, other potential fixed ranking depths (excluding 10, 20, 100, 200 and 1000) can yield results comparable to those of supervised methods across all scenarios.

Second, in scenarios balancing efficiency and effectiveness or prioritizing efficiency, distribution-based supervised RLT methods

[3] https://github.com/castorini/pyserini
[4] https://github.com/texttron/tevatron
[5] https://github.com/castorini/pygaggle
[6] https://radimrehurek.com/gensim

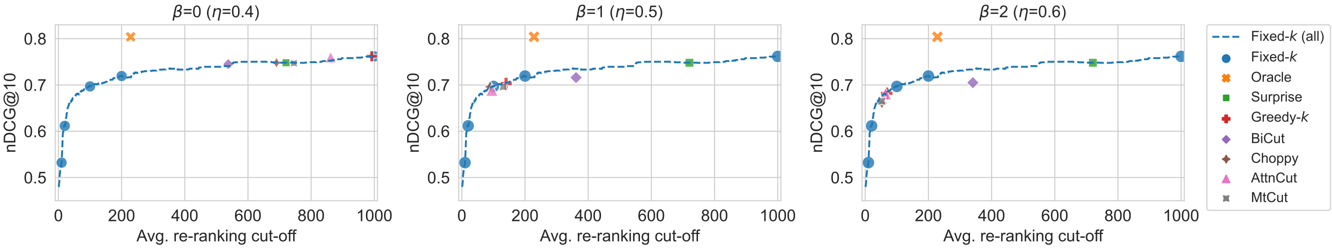


Figure 5: A comparison of RLT methods in predicting re-ranking cut-off points for BM25–RankLLaMA on TREC-DL 20. β=0 (η=0.4), β=1 (η=0.5), and β=2 (η=0.6) represent effectiveness emphasis, balance, and efficiency emphasis, respectively.

Table 1: A comparison of RLT methods in predicting re-ranking cut-off points for BM25–RankLLaMA on TREC-DL 19 and 20. Query latency is measured in seconds. The best nDCG@10 value in each column is marked in bold, and the second best is underlined. Significant differences with Fixed-k (100), Fixed-k (200) and Fixed-k (1000) are marked with *, § and †, respectively (paired t-test, $p < 0.05$).

Method	TREC-DL 19			TREC-DL 20		
	Avg. k	nDCG@10	Lat.	Avg. k	nDCG@10	Lat.
w/o re-ranking	-	$0.506^{*§†}$	-	-	$0.480^{*§†}$	-
Fixed-k (10)	10	$0.557^{*§†}$	0.30	10	$0.532^{*§†}$	0.30
Fixed-k (20)	20	$0.651^{*§†}$	0.60	20	$0.612^{*§†}$	0.60
Fixed-k (100)	100	0.730	2.98	100	$0.697^{§†}$	2.98
Fixed-k (200)	200	0.739	5.95	200	$0.719^{*†}$	5.96
Fixed-k (1000)	1000	**0.747**	29.77	1000	$\mathbf{0.762}^{*§}$	29.78
Surprise	712.12	0.743	21.20	720.56	$0.748^{*§†}$	21.46
Greedy-k (β=0)	987	0.746	29.39	992	$\mathbf{0.762}^{*§}$	29.54
BiCut (η=0.4)	385.51	0.719	11.48	537.57	$0.745^{†}$	16.01
Choppy (β=0)	842.95	<u>0.744</u>	25.10	690.24	$0.748^{*§}$	20.56
AttnCut (β=0)	904.16	**0.747**	26.92	861.87	<u>0.760</u>*§	25.67
MtCut (β=0)	843.67	0.741	25.12	745.44	$0.747^{*§†}$	22.20
Greedy-k (β=1)	242	0.737	7.21	140	$0.703^{§†}$	4.17
BiCut (η=0.5)	184.02	$0.729^{†}$	5.48	361.59	$0.716^{†}$	10.77
Choppy (β=1)	140.63	0.733	4.19	89.65	$0.696^{§†}$	2.67
AttnCut (β=1)	211.21	$0.720^{§†}$	6.29	95.11	$0.689^{§†}$	2.83
MtCut (β=1)	138.26	$0.714^{§†}$	4.12	130.98	$0.696^{†}$	3.90
Greedy-k (β=2)	242	0.737	7.21	68	$0.682^{§†}$	2.03
BiCut (η=0.6)	130.74	$0.693^{*§†}$	3.89	340.81	$0.705^{†}$	10.15
Choppy (β=2)	118.84	0.732	3.54	52.67	$0.661^{*§†}$	1.57
AttnCut (β=2)	64.16	$0.692^{*§†}$	1.91	63.89	$0.681^{§†}$	1.90
MtCut (β=2)	62.02	$0.687^{*†§}$	1.85	52.48	$0.665^{§†}$	1.56
Oracle	156.88	$0.785^{*§†}$	4.67	228.56	$0.804^{*§†}$	6.80

(Choppy, AttnCut, and MtCut) outperform the sequential labeling-based method (BiCut); however, in the scenario emphasizing effectiveness, BiCut shows a slight advantage. For instance, as shown in Figure 5, BiCut incurs lower costs to achieve nDCG@10 comparable to distribution-based methods when effectiveness is emphasized; however, in other scenarios (β=1/2 and η=0.5/0.6), the point denoting BiCut is below the dashed line denoting potential fixed re-ranking depths, while the points representing other supervised methods are on the line, indicating a worse effectiveness/efficiency balance achieved by BiCut.

Table 2: Comparison of RLT methods in predicting re-ranking cut-off points for SPLADE++–RankLLaMA on TREC-DL 19 and 20. Query latency measured in seconds. The best nDCG@10 value in each column is marked in bold, and the second best is underlined. Significant differences with Fixed-k (20) are marked with * (paired t-test, $p < 0.05$).

Method	TREC-DL 19			TREC-DL 20		
	Avg. k	nDCG@10	Lat.	Avg. k	nDCG@10	Lat.
w/o re-ranking	-	0.731^{*}	-	-	0.720^{*}	-
Fixed-k (10)	10	0.740^{*}	0.30	10	0.741^{*}	0.30
Fixed-k (20)	20	<u>0.773</u>	0.60	20	<u>0.778</u>	0.60
Fixed-k (100)	100	0.769	2.98	100	0.771	2.98
Fixed-k (200)	200	0.769	5.95	200	0.769	5.96
Fixed-k (1000)	1000	0.768	29.77	1000	0.768	29.79
Surprise	701.93	0.766	20.90	684.13	0.768	20.38
Greedy-k (β=0)	65	0.770	1.94	42	0.772	1.25
BiCut (η=0.4)	1000	0.768	29.77	1000.0	0.768	29.79
Choppy (β=0)	904.12	0.768	26.92	999.96	0.768	29.79
AttnCut (β=0)	999	0.768	29.74	999	0.768	29.76
MtCut (β=0)	999	0.768	29.74	1000.0	0.768	29.79
Greedy-k (β=1)	65	0.770	1.94	21	0.775	0.63
BiCut (η=0.5)	2	0.731^{*}	0.06	1	0.720^{*}	0.00
Choppy (β=1)	68.09	0.771	2.03	101.91	0.766	3.03
AttnCut (β=1)	46.37	0.771	1.38	53.20	0.771	1.58
MtCut (β=1)	119.67	**0.775**	3.56	996	0.768	29.67
Greedy-k (β=2)	18	0.769	0.54	21	0.775	0.63
BiCut (η=0.6)	2	0.731^{*}	0.06	1	0.720^{*}	0.00
Choppy (β=2)	1	0.731^{*}	0.00	20.57	0.764	0.61
AttnCut (β=2)	23.65	0.758^{*}	0.70	52.09	0.772	1.55
MtCut (β=2)	19.74	0.756^{*}	0.59	25.93	**0.784**	0.77
Oracle	17.44	0.810^{*}	0.52	27.69	0.820^{*}	0.82

Third, the supervised method (MtCut) learning RLT in a multi-task manner does not show a clear advantage over other supervised methods across all three trade-offs.

5.2 The impact of retriever types on RLT

To answer RQ2, we assess RLT methods (optimized for the three effectiveness/efficiency trade-offs) in the context of an LLM-based re-ranker (RankLLaMA [27]) with other novel retrievers; we explore learned sparse (SPLADE++ [17]) and dense (RepLLaMA [27]) retrievers. We present the results for SPLADE++–RankLLaMA and RepLLaMA–RankLLaMA on TREC-DL 19 and 20 in Table 2 and 3, respectively. we plot both pipelines' results for TREC-DL 20 in

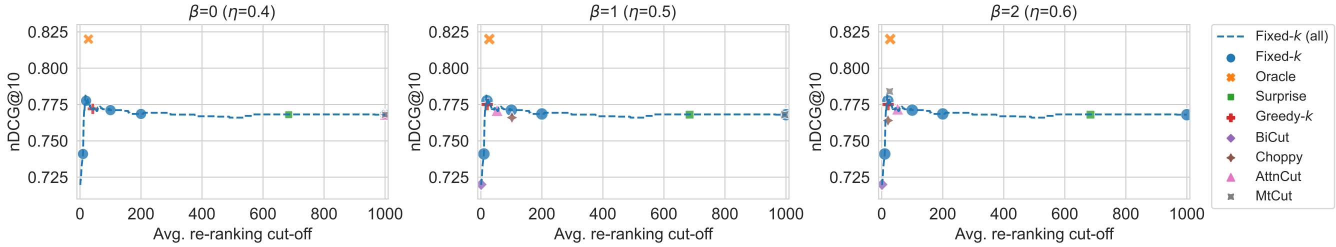


Figure 6: A comparison of RLT methods in predicting re-ranking cut-off points for Splade++–RankLLaMA on TREC-DL 20.

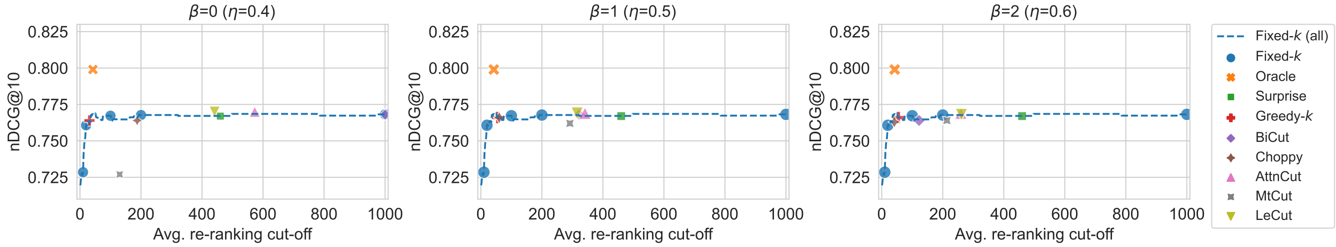


Figure 7: A comparison of RLT methods in predicting re-ranking cut-off points for RepLLaMA–RankLLaMA on TREC-DL 20.

Table 3: Comparison of RLT methods in predicting cut-off points for RepLLaMA–RankLLaMA on TREC-DL 19 and 20. Query latency measured in seconds. The best nDCG@10 value in each column is marked in bold, and the second best is underlined. Significant differences with Fixed-*k* (20) are marked with * (paired t-test, $p < 0.05$).

Method	TREC-DL 19			TREC-DL 20		
	Avg. k	nDCG@10	Lat.	Avg. k	nDCG@10	Lat.
w/o re-ranking	-	0.738*	-	-	0.720*	-
Fixed-*k* (10)	10	0.742*	0.30	10	0.729*	0.30
Fixed-*k* (20)	20	0.765	0.60	20	0.761	0.60
Fixed-*k* (100)	100	**0.769**	2.98	100	0.767	2.99
Fixed-*k* (200)	200	<u>0.768</u>	5.96	200	0.768	5.97
Fixed-*k* (1000)	1000	0.763	29.81	1000	0.768	29.86
Surprise	458.30	0.765	13.66	459.98	0.767	13.73
Greedy-*k* (β=0)	770	0.763	22.95	31	0.764	0.93
BiCut (η=0.4)	1000	0.763	29.81	1000	0.768	29.86
Choppy (β=0)	76.72	0.766	2.29	187.48	0.764	5.60
AttnCut (β=0)	928.95	0.763	27.69	572.74	**0.770**	17.10
MtCut (β=0)	509.70	0.758	15.19	130.04	0.727*	3.88
LeCut (β=0)	417.77	0.766	12.45	441.22	**0.770**	13.17
Greedy-*k* (β=1)	50	0.767	1.49	55	0.766	1.64
BiCut (η=0.5)	166.81	0.766	4.97	322.50	0.768	9.63
Choppy (β=1)	106.53	0.766	3.18	60.52	0.766	1.81
AttnCut (β=1)	458.47	0.765	13.67	341.28	<u>0.769</u>	10.19
MtCut (β=1)	582.58	0.761	17.37	291.85	0.762	8.71
LeCut (β=1)	315.37	**0.769**	9.40	315.56	<u>0.769</u>	9.42
Greedy-*k* (β=2)	50	0.767	1.49	55	0.766	1.64
BiCut (η=0.6)	153.70	0.766	4.58	122.41	0.764	3.65
Choppy (β=2)	72.07	0.766	2.15	41.43	0.763	1.24
AttnCut (β=2)	210	0.767	6.26	259.39	<u>0.769</u>	7.74
MtCut (β=2)	514.53	0.763	15.34	213.74	0.764	6.38
LeCut (β=2)	214.49	**0.769**	6.39	261.20	0.768	7.80
Oracle	23.33	0.794*	0.70	42.46	0.799*	1.27

Figure 6 and 7. Note that LeCut is only compatible with pipelines featuring a dense retriever. We have four observations.

First, different from the findings in Section 5.1, the unsupervised method Fixed-*k* (20) consistently achieves the best effectiveness/efficiency trade-off compared to supervised methods for both pipelines across all scenarios. Although some supervised methods achieve higher nDCG@10 values than Fixed-*k* (20), the nDCG@10 improvement is not statistically significant, e.g., for SPLADE++–RankLLaMA, MtCut (β=2) outperform Fixed-*k* (20) by 0.006 in terms of nDCG@10 at a comparable cost on TREC-DL 20; however, this difference is too marginal. We believe Fixed-*k* (20) performs well due to the superior retrieval capabilities of SPLADE++ and RepLLaMA. Both tend to retrieve more relevant items at the top ranks, making a shallow fixed re-ranking cutoff both effective and efficient.

Second, similar to Section 5.1, distribution-based supervised RLT methods typically achieve a better effectiveness/efficiency trade-off in re-ranking than the sequential labeling-based method (BiCut). E.g., for SPLADE++–RankLLaMA, BiCut predicts depths that are either too shallow (1 or 2) or too deep (1000); for RepLLaMA–RankLLaMA, certain distribution-based RLT methods (e.g., Choppy) achieve similar nDCG@10 values to BiCut but at a reduced re-ranking cost.

Third, different from Section 5.1, the supervised method (MtCut), which learns RLT in a multi-task manner, exhibits a superior effectiveness/efficiency trade-off compared to other methods in a specific case; however, this advantage is not consistently observed. Specifically, as shown in 6, in the scenario emphasizing efficiency, MtCut (β=2) achieves the highest nDCG@10 value (except for Oracle) for SPLADE++–RankLLaMA on TREC-DL 20 at a notably low cost. Nevertheless, as depicted in Figure 7, for RepLLaMA–RankLLaMA, the points representing MtCut consistently fall below the dashed line of Fixed-*k*, indicating a worse effectiveness/efficiency trade-off compared to other supervised methods.

Fourth, LeCut utilizes query-item embeddings from RepLLaMA to predict re-ranking cut-off points for RepLLaMA–RankLLaMA, resulting in marginal improvements over other supervised methods in nDCG@10. As shown in Table 3 and Figure 7, these improvements are often too minimal to be significant; moreover, LeCut often attains these marginal improvements at the cost of efficiency. For instance, LeCut (β=0) achieves the highest nDCG@10 value of 0.770 on TREC-DL 20, which outperforms that of Choppy (β=0) by 0.006; however, LeCut incurs more than twice the cost of Choppy.

5.3 RLT for pre-trained LM-based re-ranking

To answer **RQ3**, we evaluate RLT methods in the context of a pre-trained language model-based re-ranker (monoT5 [43]) with a lexical retriever (BM25). Note that using monoT5 [43] to re-rank the retrieved list returned by RepLLaMA [27] and Splade++ [17] yields worse results; hence we only consider the pipeline of BM25–monoT5. We report the raw result numbers on TREC-DL 19 and 20 in Table 4. We also plot the results on TREC-DL 20 in Figure 8.

Generally, the findings obtained with a pre-trained language model-based re-ranker are similar to those obtained with an LLM-based re-ranker (see Section 5.1). Specifically, (i) compared to unsupervised ones, supervised methods only show an advantage in the scenario emphasizing effectiveness, where supervised methods achieve better re-ranking performance at a lower cost; e.g., BiCut (η=0.4) achieves an nDCG@10 value comparable to that of Fixed-k (1000) while incurring only half the cost on TREC-DL 20; however, similar to Section 5.1, potential fixed re-ranking depths (excluding 10, 20, 100, 200, and 1000) can still yield results that are very similar to those obtained with supervised methods (see Figure 8); (ii) distribution-based supervised methods perform better in scenarios balancing efficiency and effectiveness or prioritizing efficiency, while the sequential labeling-based method (BiCut) shows a slight advantage in the scenario emphasizing effectiveness; and (iii) the supervised method (MtCut), which learns RLT in a multi-task manner, does not consistently show a clear advantage over other supervised methods across all three trade-offs.

5.4 Error analysis

To understand the reasons behind the less-than-ideal performance of supervised RLT methods, we compare distributions of re-ranking cut-off points predicted by Oracle with those predicted by a supervised method. We consider two relatively effective methods,[7] MtCut (β=2) and Choopy (β=2), for pipelines featuring novel retrievers, SPLADE++ and RepLLaMA on TREC-DL 20; see Figure 9. First, both methods fail to predict a re-ranking cut-off of zero. For both pipelines, around 20% of queries do not need re-ranking. Thus, enhancing supervised RLT methods' ability to predict when re-ranking is unnecessary is crucial. Second, both methods perform worse when truncating RepLLaMA's retrieved lists (see Figure 9c and 9d) compared to SPLADE++ (see Figure 9a and 9b). Especially, Choppy (β=2) seems to underfit for RepLLaMA (see 9d), suggesting a potential need for more training data.

Table 4: A comparison of RLT methods in predicting re-ranking cut-off points for BM25–monoT5 on TREC-DL 19 and 20. Query latency measured in seconds. The best nDCG@10 value in each column is marked in bold, and the second best is underlined. Significant differences with Fixed-k (100), Fixed-k (200) and Fixed-k (1000) are marked with *, § and †, respectively (paired t-test, $p < 0.05$).

Method	TREC-DL 19			TREC-DL 20		
	Avg. k	nDCG@10	Lat.	Avg. k	nDCG@10	Lat.
w/o re-ranking	-	$0.506^{*\S\dagger}$	-	-	$0.480^{*\S\dagger}$	-
Fixed-k (10)	10	$0.553^{*\S\dagger}$	0.14	10	$0.523^{*\S\dagger}$	0.14
Fixed-k (20)	20	$0.634^{*\S\dagger}$	0.27	20	$0.604^{*\S\dagger}$	0.27
Fixed-k (100)	100	0.706	1.37	100	$0.676^{\dagger}$	1.37
Fixed-k (200)	200	0.717	2.73	200	$0.684^{\dagger}$	2.73
Fixed-k (1000)	1000	**0.730**	13.66	1000	$\mathbf{0.713}^{*\S}$	13.66
Surprise	712.12	0.721	9.73	720.56	$0.705^{*\S}$	9.84
Greedy-k (β=0)	993	**0.730**	13.57	991	$\mathbf{0.713}^{*\S}$	13.54
BiCut (η=0.4)	385.51	0.702	5.27	537.57	$0.710^{*\S}$	7.34
Choppy (β=0)	783.56	0.727	10.70	865.56	$0.711^{*\S}$	11.82
AttnCut (β=0)	888.02	$\underline{0.729}$	12.13	931.26	$\underline{0.712}^{*\S}$	12.72
MtCut (β=0)	791.42	0.722	10.81	756.74	$\underline{0.712}^{*\S}$	10.34
Greedy-k (β=1)	112	0.709	1.53	162	$0.678^{\dagger}$	2.21
BiCut (η=0.5)	184.02	0.711	2.51	361.59	0.697	4.94
Choppy (β=1)	161.37	0.714	2.20	203.17	$0.675^{\dagger}$	2.78
AttnCut (β=1)	197.91	0.701	2.70	112.83	$0.661^{\dagger}$	1.54
MtCut (β=1)	144.51	$0.690^{\S\dagger}$	1.97	131.06	0.681	1.79
Greedy-k (β=2)	112	0.709	1.53	86	$0.674^{\dagger}$	1.17
BiCut (η=0.6)	130.74	$0.672^{*\S\dagger}$	1.79	340.81	0.690	4.66
Choppy (β=2)	116.72	0.711	1.59	110.56	$0.669^{\dagger}$	1.51
AttnCut (β=2)	67.53	$0.677^{*\S\dagger}$	0.92	72.94	$0.663^{\dagger}$	1.00
MtCut (β=2)	62.02	$0.668^{*\S\dagger}$	0.85	53.56	$0.652^{\dagger}$	0.73
Oracle	180.91	$0.774^{*\S\dagger}$	2.47	213.67	$0.768^{*\S\dagger}$	2.92

6 RELATED WORK

Ranked list truncation. Ranked list truncation (RLT) is also known as query cut-off prediction [10, 23]. For a query and a ranked list of documents, the RLT task is to predict the number of items in the ranked list that should be returned, to optimize a user-defined metric [7]. The task can potentially benefit IR applications where it is money- and time-consuming to review a returned item, e.g., in patent search [26] and legal search [54, 57]. Early work is mainly assumption-based (hence non-neural-based). This kind of research focuses on modeling score distributions by fitting prior distributions to them [3, 32], which helps identify the best cut-off. However, prior assumptions on score distributions do not always hold as retrieval settings change [25, 57]; hence we do not study this line of studies in our work. Assumption-free methods, on the other hand, learn to predict the truncation position during training and do not rely on a prior assumption. We have already introduced those methods (BiCut, Choppy, AttnCut, MtCut and LeCut) in Section 4.2.

Zamani et al. [62] apply the RLT method from [6] to truncate retrieval lists returned by BM25 for BERT-based re-ranking [42], finding that truncating the retrieval result list to avoid including a large number of non-relevant items in the lower ranks, achieves better re-ranking performance than using fixed cut-offs for all queries.

We differ from Zamani et al. [62] as we provide a systematic and comprehensive study into the use of RLT methods in the context of re-ranking, especially newly emerged LLMs-based re-ranking.

Improving neural re-ranking efficiency. Improving neural re-ranking efficiency has been extensively studied. There are two ideas to improve the efficiency [19]: (i) speed up inference of a neural re-ranker, and (ii) reducing the number of inferences of a neural re-ranker. Approaches to (i) include a simpler re-ranker model [20],

[7] Due to space limitations, we provide error analysis for all methods in our repository.

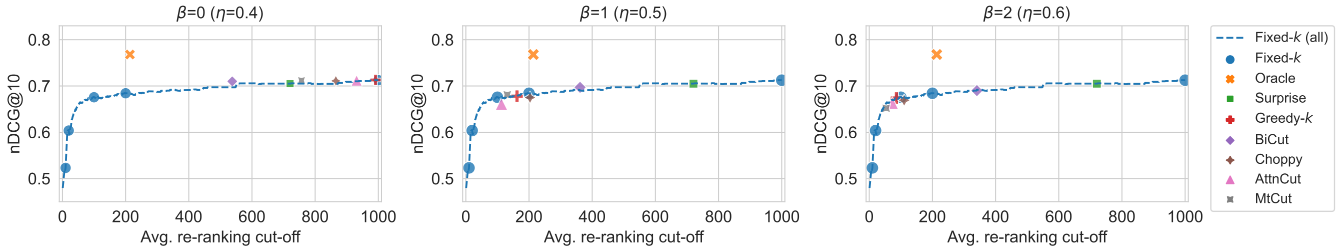
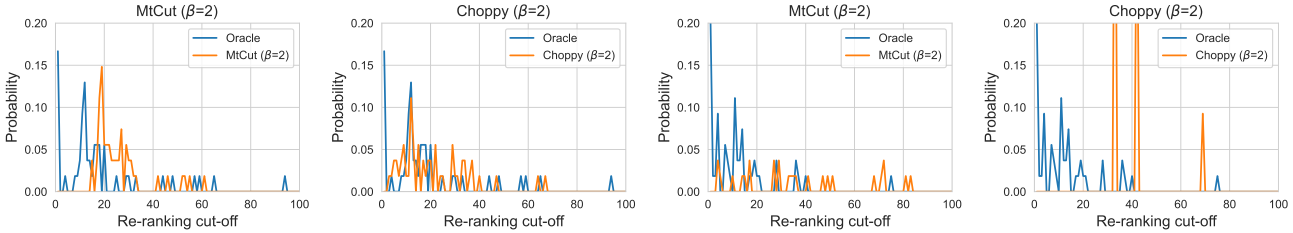


Figure 8: A comparison of RLT methods in predicting re-ranking cut-off points for BM25–monoT5 on TREC-DL 20.



(a) SPLADE++–RankLLaMA **(b) SPLADE++–RankLLaMA** **(c) RepLLaMA–RankLLaMA** **(d) RepLLaMA–RankLLaMA**

Figure 9: The distribution of re-ranking cut-off points on TREC-DL 20.

distilling knowledge in BERT [15] into a smaller re-ranker [18], pre-computing item representations at indexing time [30], and early-exiting [51, 61]. Studies into (ii) are more related to our work. It includes *multi-stage re-ranking* [33, 44, 59, 64] and *candidate pruning* [13, 24, 44].

Multi-stage re-ranking first exploits faster and less effective re-rankers to discard likely non-relevant items and sends fewer candidate items to more expensive re-rankers in later stages. E.g., Zhang et al. [64] first use a feature-based LtR model to reorder the items returned by BM25 and then send the top-*k* (applied to all queries) items returned by the faster re-ranker to a BERT re-ranker.

Candidate pruning trims the candidate list in the first (or earlier) stage and then forwards the pruned ranked list to the next stage re-ranking. Wang et al. [59] propose a boosting algorithm for jointly learning pruning and ranker stages. Culpepper et al. [13] use a cascade of binary classifiers based on random forests; each classifier is used to predict whether to truncate the given ranked list at a specific cut-off value. Li et al. [24] propose a score-thresholding method, which makes sure the trimmed candidate list produces re-ranking outcomes that satisfy the user-specified error tolerance of an IR evaluation metric.

We also differ from Asadi and Lin [4] and Tonellotto et al. [55], who investigate improving the efficiency of candidate generation, i.e., first-stage retrieval. Specifically, Tonellotto et al. [55] predict the number of candidate items that should be retrieved by the candidate generation algorithm WAND [8] on a per-query basis. Our focus lies in improving re-ranking efficiency by truncating retrieved lists; in our setup, the retriever always returns a fixed number of items.

7 CONCLUSIONS & FUTURE WORK

We have reproduced numerous RLT methods in the "*retrieve-then-re-rank*" setup. We showed that findings on RLT do not generalize well to this new setup. We found that (i) supervised RLT methods do not demonstrate a clear advantage over their unsupervised counterparts; potential fixed re-ranking depths can closely approximate the effectiveness/efficiency trade-off achieved by supervised methods; (ii) distribution-based supervised methods achieve better effectiveness/efficiency trade-offs than their sequential labeling-based counterpart in most cases; the latter attains better re-ranking effectiveness at a lower cost for pipelines using BM25 retrieval; (iii) jointly learning RLT with other tasks [57] does not consistently yield a clear improvement; it only demonstrate a superior re-ranking effectiveness/efficiency trade-off for SPLADE++–RankLLaMA; and (iv) incorporating neural retriever embeddings [29] does not exhibit a clear advantage; it merely yields marginal improvements in re-ranking effectiveness for RepLLaMA–RankLLaMA.

We also learn valuable lessons from our experiments: (i) the type of retriever significantly affects RLT for re-ranking; with an effective retriever like SPLADE++ or RepLLaMA, a fixed re-ranking depth of 20 can already yield an excellent effectiveness/efficiency trade-off; increasing the fixed depth do not significantly improve effectiveness; using a fixed depth of 200 for retrieved lists returned by RepLLaMA, as done by Ma et al. [27], results in unnecessary computational costs; and (ii) the type of re-ranker (LLM or pre-trained LM-based) does not appear to influence the findings.

We identify future directions: (i) the gap between Oracle and RLT methods highlights the necessity of enhancing RLT methods for re-ranking; we plan to solve the potential data scarcity issue highlighted in Section 5.4, and explore the use of query performance prediction (QPP) methods for predicting query-specific re-ranking cut-offs [2, 34, 35, 37, 38]; (ii) we only consider point–wise re-rankers; we plan to explore RLT for pair-wise and list-wise LLM-based re-rankers [46–48, 63]; and (iii) we plan to explore RLT for re-ranking in conversational search (CS) [1, 36, 39–41, 52].

Acknowledgments. We thank Yiding Liu (Baidu Inc.) for insightful discussions that contributed to the development of our ideas. This research was supported by the China Scholarship Council (CSC) under grant number 202106220041, the Dutch Research Council (NWO), under project numbers 024.004.022, NWA.1389.20.183, and KICH3.LTP.20.006, and the European Union's Horizon Europe program under grant agreement No 101070212.

REFERENCES

[1] Zahra Abbasiantaeb, Chuan Meng, David Rau, Antonis Krasakis, Hossein A Rahmani, and Mohammad Aliannejadi. 2023. LLM-based Retrieval and Generation Pipelines for TREC Interactive Knowledge Assistance Track (iKAT) 2023. In *TREC*.

[2] Negar Arabzadeh, Chuan Meng, Mohammad Aliannejadi, and Ebrahim Bagheri. 2024. Query Performance Prediction: From Fundamentals to Advanced Techniques. In *ECIR*. Springer, 381–388.

[3] Avi Arampatzis, Jaap Kamps, and Stephen Robertson. 2009. Where to Stop Reading a Ranked List? Threshold Optimization using Truncated Score Distributions. In *SIGIR*. 524–531.

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

[5] Arian Askari, Roxana Petcu, Chuan Meng, Mohammad Aliannejadi, Amin Abolghasemi, Evangelos Kanoulas, and Suzan Verberne. 2024. Self-seeding and Multi-intent Self-instructing LLMs for Generating Intent-aware Information-Seeking dialogs. *arXiv preprint arXiv:2402.11633* (2024).

[6] Dara Bahri, Yi Tay, Che Zheng, Donald Metzler, and Andrew Tomkins. 2020. Choppy: Cut Transformer for Ranked List Truncation. In *SIGIR*. 1513–1516.

[7] Dara Bahri, Che Zheng, Yi Tay, Donald Metzler, and Andrew Tomkins. 2023. Surprise: Result List Truncation via Extreme Value Theory. In *SIGIR*. 2404–2408.

[8] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient Query Evaluation using a Two-Level Retrieval Process. In *CIKM*. 426–434.

[9] Sebastian Bruch, Claudio Lucchese, and Franco Maria Nardini. 2023. Efficient and Effective Tree-based and Neural Learning to Rank. *Foundations and Trends in Information Retrieval* 17, 1 (2023), 1–123.

[10] Daniel Cohen, Bhaskar Mitra, Oleg Lesota, Navid Rekabsaz, and Carsten Eickhoff. 2021. Not All Relevance Scores are Equal: Efficient Uncertainty and Calibration Modeling for Deep Retrieval Models. In *SIGIR*. 654–664.

[11] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *TREC*.

[12] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2019. Overview of the TREC 2019 Deep Learning Track. In *TREC*.

[13] J Shane Culpepper, Charles LA Clarke, and Jimmy Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *Proceedings of the 21st Australasian Document Computing Symposium*. 17–24.

[14] Donald A Darling. 1957. The Kolmogorov-Smirnov, Cramer-Von Mises Tests. *The Annals of Mathematical Statistics* 28, 4 (1957), 823–838.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*. 4171–4186.

[16] Andrew Drozdov, Honglei Zhuang, Zhuyun Dai, Zhen Qin, Razieh Rahimi, Xuanhui Wang, Dana Alon, Mohit Iyyer, Andrew McCallum, Donald Metzler, et al. 2023. PaRaDe: Passage Ranking using Demonstrations with LLMs. In *Findings of EMNLP*. 14242–14252.

[17] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *SIGIR*. 2353–2359.

[18] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Understanding BERT Rankers Under Distillation. In *SIGIR*. 149–152.

[19] Lukas Gienapp, Maik Fröbe, Matthias Hagen, and Martin Potthast. 2022. Sparse Pairwise Re-ranking with Pre-trained Transformers. In *ICTIR*. 72–80.

[20] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2020. Interpretable & Time-Budget-Constrained Contextualization for Re-Ranking. In *ECAI 2020 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain-Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. IOS Press, 1–8.

[21] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[22] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *ICML*. PMLR, 1188–1196.

[23] Oleg Lesota, Navid Rekabsaz, Daniel Cohen, Klaus Antonius Grasserbauer, Carsten Eickhoff, and Markus Schedl. 2021. A Modern Perspective on Query Likelihood with Deep Generative Retrieval Models. In *ICTIR*. 185–195.

[24] Minghan Li, Xinyu Zhang, Ji Xin, Hongyang Zhang, and Jimmy Lin. 2022. Certified Error Control of Candidate Set Pruning for Two-Stage Relevance Ranking. In *EMNLP*. 333–345.

[25] Yen-Chieh Lien, Daniel Cohen, and W Bruce Croft. 2019. An Assumption-Free Approach to the Dynamic Truncation of Ranked Lists. In *ICTIR*. 79–82.

[26] Mihai Lupu and Allan Hanbury. 2013. Patent Retrieval. *Foundations and Trends in Information Retrieval* 7, 1 (2013), 1–97.

[27] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2023. Fine-Tuning LLaMA for Multi-Stage Text Retrieval. *arXiv preprint arXiv:2310.08319* (2023).

[28] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-Shot Listwise Document Reranking with a Large Language Model. *arXiv preprint arXiv:2305.02156* (2023).

[29] Yixiao Ma, Qingyao Ai, Yueyue Wu, Yunqiu Shao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2022. Incorporating Retrieval Information into the Truncation of Ranking Lists for Better Legal Search. In *SIGIR*. 438–448.

[30] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Efficient Document Re-Ranking for Transformers by Precomputing Term Representations. In *SIGIR*. 49–58.

[31] Sean MacAvaney, Nicola Tonellotto, and Craig Macdonald. 2022. Adaptive Re-Ranking with a Corpus Graph. In *CIKM*. 1491–1500.

[32] Raghavan Manmatha, Toni Rath, and Fangfang Feng. 2001. Modeling Score Distributions for Combining the Outputs of Search Engines. In *SIGIR*. 267–275.

[33] Yoshitomo Matsubara, Thuy Vu, and Alessandro Moschitti. 2020. Reranking for Efficient Transformer-based Answer Selection. In *SIGIR*. 1577–1580.

[34] Chuan Meng. 2024. Query Performance Prediction for Conversational Search and Beyond. In *SIGIR*.

[35] Chuan Meng, Mohammad Aliannejadi, and Maarten de Rijke. 2023. Performance Prediction for Conversational Search Using Perplexities of Query Rewrites. In *QPP++2023*. 25–28.

[36] Chuan Meng, Mohammad Aliannejadi, and Maarten de Rijke. 2023. System Initiative Prediction for Multi-turn Conversational Information Seeking. In *CIKM*. 1807–1817.

[37] Chuan Meng, Negar Arabzadeh, Mohammad Aliannejadi, and Maarten de Rijke. 2023. Query Performance Prediction: From Ad-hoc to Conversational Search. In *SIGIR*. 2583–2593.

[38] Chuan Meng, Negar Arabzadeh, Arian Askari, Mohammad Aliannejadi, and Maarten de Rijke. 2024. Query Performance Prediction using Relevance Judgments Generated by Large Language Models. *arXiv preprint arXiv:2404.01012* (2024).

[39] Chuan Meng, Pengjie Ren, Zhumin Chen, Christof Monz, Jun Ma, and Maarten de Rijke. 2020. RefNet: A Reference-aware Network for Background Based Conversation. In *AAAI*.

[40] Chuan Meng, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tengxiao Xi, and Maarten de Rijke. 2021. Initiative-Aware Self-Supervised Learning for Knowledge-Grounded Conversations. In *SIGIR*. 522–532.

[41] Chuan Meng, Pengjie Ren, Zhumin Chen, Weiwei Sun, Zhaochun Ren, Zhaopeng Tu, and Maarten de Rijke. 2020. DukeNet: A Dual Knowledge Interaction Network for Knowledge-Grounded Conversation. In *SIGIR*. 1151–1160.

[42] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[43] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *EMNLP*. 708–718.

[44] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-Stage Document Ranking with BERT. *arXiv preprint arXiv:1910.14424* (2019).

[45] James Pickands III. 1975. Statistical Inference Using Extreme Order Statistics. *the Annals of Statistics* (1975), 119–131.

[46] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. *arXiv preprint arXiv:2309.15088* (2023).

[47] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! *arXiv preprint arXiv:2312.02724* (2023).

[48] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. 2023. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. *arXiv preprint arXiv:2306.17563* (2023).

[49] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3, 4 (2009), 333–389.

[50] Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving Passage Retrieval with Zero-Shot Question Generation. In *EMNLP*. 3781–3797.

[51] Luca Soldaini and Alessandro Moschitti. 2020. The Cascade Transformer: an Application for Efficient Answer Sentence Selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 5697–5708.

[52] Weiwei Sun, Chuan Meng, Qi Meng, Zhaochun Ren, Pengjie Ren, Zhumin Chen, and Maarten de Rijke. 2021. Conversations Powered by Cross-Lingual Knowledge. In *SIGIR*. 1442–1451.

[53] Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agent. In *EMNLP*. 14918–14937.

[54] Stephen Tomlinson, Douglas W Oard, Jason R Baron, and Paul Thompson. 2007. Overview of the TREC 2007 Legal Track.. In *TREC*.

[55] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and Effective Retrieval using Selective Pruning. In *WSDM*. 63–72.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NeurIPS*. 5998–6008.

[57] Dong Wang, Jianxin Li, Tianchen Zhu, Haoyi Zhou, Qishan Zhu, Yuxin Wen, and Hongming Piao. 2022. MtCut: A Multi-Task Framework for Ranked List

Truncation. In *WSDM*. 1054–1062.

[58] Lidan Wang, Jimmy Lin, and Donald Metzler. 2010. Learning to Efficiently Rank. In *SIGIR*. 138–145.

[59] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *SIGIR*. 105–114.

[60] Chen Wu, Ruqing Zhang, Jiafeng Guo, Yixing Fan, Yanyan Lan, and Xueqi Cheng. 2021. Learning to Truncate Ranked Lists for Information Retrieval. In *AAAI*, Vol. 35. 4453–4461.

[61] Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020. Early Exiting BERT for Efficient Document Ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. 83–88.

[62] Hamed Zamani, Michael Bendersky, Donald Metzler, Honglei Zhuang, and Xuanhui Wang. 2022. Stochastic Retrieval-Conditioned Reranking. In *ICTIR*. 81–91.

[63] Xinyu Zhang, Sebastian Hofstätter, Patrick Lewis, Raphael Tang, and Jimmy Lin. 2023. Rank-without-GPT: Building GPT-Independent Listwise Rerankers on Open-Source Large Language Models. *arXiv preprint arXiv:2312.02969* (2023).

[64] Yue Zhang, ChengCheng Hu, Yuqi Liu, Hui Fang, and Jimmy Lin. 2021. Learning to Rank in the Age of Muppets: Effectiveness–Efficiency Tradeoffs in Multi-Stage Ranking. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*. 64–73.

[65] Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Berdersky. 2023. Beyond Yes and No: Improving Zero-Shot LLM Rankers via Scoring Fine-Grained Relevance Labels. *arXiv preprint arXiv:2310.14122* (2023).

[66] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. 2023. Open-source Large Language Models are Strong Zero-shot Query Likelihood Models for Document Ranking. *arXiv preprint arXiv:2310.13243* (2023).

[67] Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2023. A Setwise Approach for Effective and Highly Efficient Zero-shot Ranking with Large Language Models. *arXiv preprint arXiv:2310.09497* (2023).