Tournament Sorting: Effective and Robust Setwise Approach for Passage Reranking with Open-source Large Language Model

Submitted as Research Report / Honours / Master Dissertation in SIT723

SUBMISSION DATE

T3-2024

Quang Huynh

STUDENT ID s224352548

COURSE - Master of Applied Artificial Intelligence (Professional) (S737)

Supervised by: Dr. Bahadorreza Ofoghi

Abstract

The increasing reliance on large language models (LLMs) for the classification of tasks in text retrieval systems in recent years has highlighted significant limitations, particularly in their dependence on proprietary architectures and the challenges of addressing positional bias of first retrieval. These challenges hinder reproducibility and broader adoption in open source applications, necessitating solutions that balance effectiveness and accessibility. This study aims to explore the potential of open source LLMs for zero-shot list-wise and set-wise reranking, focusing on overcoming critical obstacles outlined above. A novel tournament sorting algorithm is introduced to address positional bias, ensuring robust and effective reranking outcomes. Comparative evaluations were conducted against SOTA zero-shot listwise and setwise methods, and proprietary and fine-tuned models, using benchmark datasets such as TREC COVID and TREC DL. The results demonstrate that the proposed approach achieves competitive performance in NDCG and MRR metrics while maintaining resilience to variations in initial rankings. This research underscores the viability of open-source models for achieving high-accuracy text retrieval, providing a pathway for accessible and reproducible advancements in the field.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Aim & Objectives

Text retrieval is an essential component of many NLP tasks, particularly in open-domain question answering, where the system must identify and analyse a document to generate a response to a query. A typical QA pipeline consists of two key stages: retrieval and reranking. In the retrieval stage, the system selects a set of k relevant documents or passages from a large corpus. These k candidates are then re-evaluated in the reranking stage using a more advanced and computationally intensive method to ensure higher accuracy.

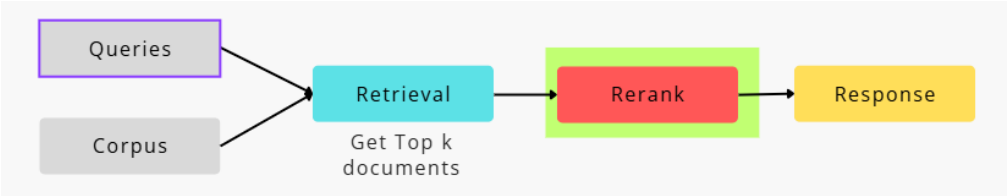

Figure 1: Text retrieval in QA system. The system starts with user queries and a corpus of documents as inputs, followed by a retrieval and reranking process, finally generates a response based on the reranked results.

LLMs have recently demonstrated remarkable effectiveness and efficiency in the reranking stage. In the context of prompting method, the application of LLMs can be broadly categorized into three primary approaches: Pointwise [2] [14] [3], Pairwise [16] [24], and Listwise [33] [29] [27].

The listwise approach has demonstrated both effectiveness and efficiency by optimizing the relevance of an entire ranked list of documents in response to a given query. Recent studies have reported SOTA performance with the listwise approach; however, several critical challenges remain. Much of the previous research concentrated on proprietary LLMs, such as GPT-3 and GPT-4, for inference tasks [29]. While these models have achieved impressive results, their reliance poses serious limitations. Proprietary models are often non-reproducible due to closed architectures, restricted accessibility, and licensing constraints, which impede independent validation of research findings.

Some studies have attempted to fine-tune open-source models while utilizing GPT-based proprietary systems exclusively for annotation [27] [28]. Although this approach addresses some concerns, it still reflects a significant reliance on commercial systems. Other works have leveraged the MSMARCO dataset [20] [36] [34] as a training resource; however, this dataset is primarily suited for pointwise objectives, as it typically contains only one labeled passage per query on average [32]. This limitation reduces its applicability for listwise training approaches, which require richer relevance annotations to optimize ranking

performance effectively. Another persistent challenge in listwise reranking is positional bias. The performance of reranking models is often highly dependent on the initial ranking of retrieved passages.

In this research, we aim to develop a zero-shot listwise reranking system that fully leverages open-source LLMs. Additionally, we seek to address a notable gap in the field by conducting comprehensive evaluations that directly compare the performance of proprietary models against open-source alternatives. Such comparisons are essential for understanding the practical trade-offs and determining the viability of open-source solutions for real-world applications.

To achieve this, we will employ a setwise architecture that outputs only the top candidate for each prompt, rather than producing a fully ordered list as is typical in listwise reranking systems. Most importantly, we propose a novel tournament sorting method to address positional bias, which remains a critical challenge in enhancing the robustness and generalizability of listwise approaches across benchmark dataset.

Furthermore, we will evaluate our approach alongside other SOTA setwise sorting algorithms to investigate the effectiveness of sorting algorithms in general in resolving the "lost in the middle" problem, where highly relevant passages may be overlooked due to suboptimal ranking positions [13]. To this end, our study focuses on the following research questions:

- RQ 1: How does the performance of open-source models in zero-shot listwise reranking compared to proprietary models and their distil/fine-tuned model?
- RQ 2: Does different list-wise prompting and sorting algorithm solve initial positional bias?

1.2 Structure

The paper consists of five main sections. The first section outlines the research goal, objectives, and research questions. Next, we present the literature review, discussing SOTA performance in passage reranking. In Section 3, we introduce the methodology of the proposed solution, followed by the experimental setup and artifact development. We then present and analyze the results, including findings from several ablation studies. Finally, we conclude the paper by discussing the project's sustainability, limitations, and potential directions for future work.

2 Literature Review

Using the re-ranking phase, previous research has attempted to maximize the relevance and accuracy of the ranked result. BM25 [25] is one of the first sparse retrievers that relies on traditional term-based representations of text to calculate the relevance score based on the frequency of the term and the inverse frequency in all documents. However, its main limitation is the lack of contextual understanding, as it cannot capture semantic relationships between words.

To address this challenge, dense retrieval methods have been employed to capture the semantic meaning of text. These methods utilize dense, continuous vector representations of both queries and passages to identify relevant information. Cross-encoders such as BERT [26] have proven effective in generating embeddings that model the relationship between queries and passages. In contrast, ColBERT [9] maintains separate interactions between query and document features, deferring the query-document interaction until the output layer. This approach preserves the "query-document decoupling" architecture, enabling the document or item embeddings to be frozen post-training. As a result, these embeddings can be stored in an index and retrieved during inference.

Additionally, Contriever [8], which uses contrastive learning, fine-tunes the representations of queries and passages to effectively distinguish relevant from irrelevant information. Furthermore, a novel loss function based on coverage and overlap scores in a triple-assisted learning framework has also shown promise in improving dense retrieval results [17]. However, generating dense embeddings for large corpus requires significant memories and computational power compared to sparse retrieval.

To handle this, a method like SPLADE [4] has employed a hybrid approach, combining sparse and dense representations to achieve high retrieval performance and computational efficiency. Another study, such as COIL [6] incorporates contextualized word representations, such as those from BERT or similar into the inverted list structure. This enables the system to better capture nuances in meaning while still maintains an efficient exact lexical match, which is important for fast retrieval and scalability in large document collections.

LLMs have recently demonstrated remarkable effectiveness and efficiency, both zero-shot capabilities and fine-tuned models. Prior research highlights their competitive performance, and often rivaling dense retrieval methods. In the context of the prompting method, the application of LLMs can be broadly categorized into three primary approaches: pointwise, pairwise, and listwise.

In the *pointwise* reranking, each passage is independently scored based on its relevance

to the query. Recent work, such as UPR [2], generates a query for each passage and takes advantage of the likelihood of the input question to compute the relevance score. Other works, including InPars [14], have employed a few-shot approach with methods to generate synthetic data for information retrieval tasks. Additionally, PaRaDe [3] proposes a difficulty-based selection (DBS) method to identify challenging few-shot examples to include in the prompt. Likewise, HyDE [7] leverages LLM to generate a hypothetical document for an unsupervised dense encoder. However, the biggest issue with these methods is that they focus on binary classification instead of directly optimizing ranking metrics, making them lag behind in terms of effectiveness. To modify this situation, [35] has introduced fine-grained relevance labels in LLM, for example 'Highly Relevant', 'Somewhat Relevant', 'Not Relevant', and collect their likelihood scores to estimate the ranking score. However, this method and all other point-wise approaches treat each document or passage individually and predict its relevance score in isolation. This means that the model does not explicitly learn or account for the relative order between documents, leading to suboptimal ranking performance.

Pairwise methods, rather than predicting the relevance of individual documents in isolation, introduce relative order between pairs of documents. Recent pairwise approaches, including ASAP [16] and PRP [24], have demonstrated superior effectiveness. However, scalability remains a challenge, as generating all possible pairs of documents can be computationally expensive. Recent works, such as those of [24] and [37], have proposed sorting algorithms to improve efficiency, reducing the time complexity from quadratic to logarithmic.

The *listwise* approach has emerged as a balanced alternative to *pointwise* and *pairwise* methods, offering both effectiveness and efficiency by evaluating and optimizing the relevance of an entire list of documents in response to a query [33]. Notable examples include RankGPT [29] and RankVicuna [27], which utilize instructional permutation generation to output passage identifiers in descending ranking order. Furthermore, methods such as variable sliding windows in Rank-Zephyr [28] and passage embedding prompting in PE-Rank [23] have been proposed to address token limitations in LLMs and reduce suboptimal ranking in each sliding window.

However, these approaches often rely on proprietary models for content generation objectives rather than specifically learning-to-rank. Furthermore, GPT models often produce non-deterministic outputs and struggle to consistently adhere to the specified format in their responses. To address this, studies such as FIRST [5] have explored accessing token logits to retrieve the likelihood of passage identifiers for reranking, thus mitigating unexpected output issues observed in commercial models such as GPT [28]. Additional efforts have used GPT-4 for annotation and have fine-tuned or distilled into smaller, specialized models [29]. Although these methods achieve SOTA performance, they

heavily depend on commercial models like GPT-4 as teachers for fine-tuning, which incurs significant computational cost due to extensive training hours. To solve this problem, several studies made an effort to create new retrieval datasets that are more suitable for listwise reranking, such as NovalEval [29], or leverage BM25 and fine-tuned Contriever as silver ranking to approximate gold ranking [32]. These datasets not only facilitate the development of listwise rerankers but also help mitigate potential risks associated with data contamination, which can lead to overestimation of LLM performance and introduce bias [21].

Another limitation of the listwise approach is positional bias, often referred to as the "lost in the middle" problem [13], where the re-ranking results are overly influenced by the initial retrieval steps. Despite efforts to mitigate this bias, such as those of [30] and [37], these solutions require multiple LLM calls to shuffle the order, making them more expensive and challenging compared to point-wise or pairwise approaches.

Hence, leveraging the capabilities of LLMs for listwise reranking remains a complex and underexplored challenge, since recent studies depend on a commercial model for the zero-shot and fine-tuning approach.

3 Research Design & Methodology

3.1 Listwise vs Setwise Rerankers

For a user query q and a set of candidate passages $\{P_1, P_1, ..., P_n\}$ obtained from a corpus or a retrieval stage, a listwise reranker processes the query q along with all passages simultaneously and returns a reordered list of input passage identifiers (Figure 2). Two common approaches to prompt LLMs to rank documents list-wise are generation [33] and likelihood [5]. In the generation approach, the method involves prompting the language model to produce a response or text based on the query and candidate passages (e.g. [2] > [3] > [1]). The original listwise method relies on the LLM's next-token generation capability to produce a complete ordered list of document labels. However, this can sometimes result in outputs with unexpected formats, which is commonly observed in proprietary models such as GPT [27]. In contrast, the likelihood approach assesses the relevance of a document by calculating the logits of the passage given the query under the model's distribution. Then, all the passage identifiers are concatenated in the output in a non-ascending order format [6].

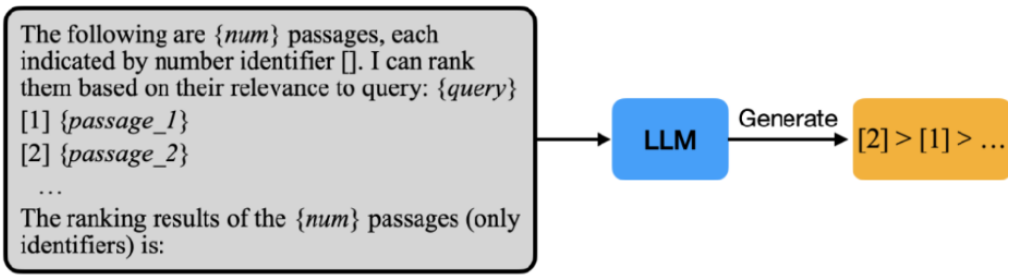

Figure 2: Listwise prompting method in generation mode. The prompt includes a sublist of passages and output a complete ordered list of document labels

In contrast to listwise reranker, setwise approach takes the query q and a set of candidate passages $\{P_1, P_1, ..., P_n\}$ as input and returns only the most relevant passage identifier (Figure 3). Similar to the listwise method, this is achieved by generation or assessing the likelihood of each passage label being selected as the most relevant. The key difference is that the setwise approach outputs only the top 1 or 2 passages in a sliding window, rather than the entire list of passages as in the listwise method. The outputs are then reordered using sorting algorithms, which will be discussed in the following sections. As shown in a recent article [37], the greatest advantage of this method is that it is more robust to variations in initial classification order compared to listwise candidates.

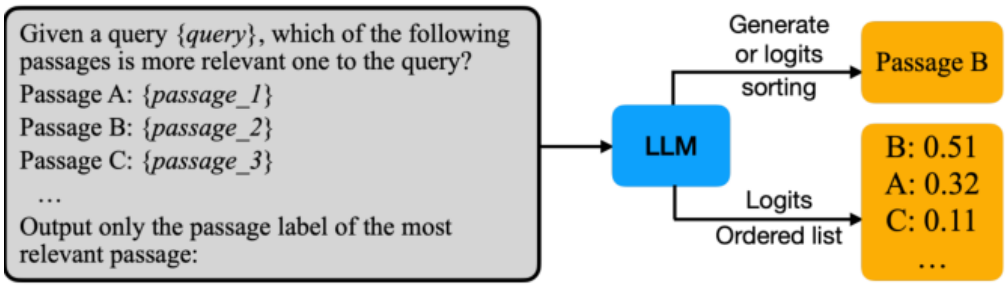

Figure 3: Setwise prompting method in generation and likelihood mode. The prompt includes a subist of passages and output the most relevant passage

3.2 Prompt Design

Inspired by [37], our method uses a simple instruction template to prompt the LLM to perform zero-shot setwise ranking. Also, we will only use the likelihood method in this phase to avoid unexpected behavior observed in the generation method:

```
Given a query {query}, which of the following passages is more relevant one to
the query?
[1]: {passage_1}
[2]: {passage_2}
[3]: {passage_3}
.....
Output only the passage label of the most relevant passage:
```

Target query is presented, followed by line-by-line candidate passages and a specific instruction at the end.

For a fair comparison of likelihood *listwise* reranking, we also use the same prompting method for baseline although we concatenate all passage identifiers at the end, separating by ">". For the generation approach, we use the permutation approach in RankGPT as follows:

```
You are RankGPT, an intelligent assistant that can rank passages based on
their relevancy to the query. I will provide you with {num} passages, each
indicated by number identifier [] Rank the passages based on their relevance
to query: {query}
[1]: {passage_1}
[2]: {passage_2}
[3]: {passage_3}
.....
The search query is: {query}
I will rank the {{num}} passages above based on their relevance to the search query.
```

```
The passages will be listed in descending order using identifiers, and the most
relevant passages should be listed first, and the output format should be
[] > [] > etc, e.g., [1] > [2] > etc.

The ranking results of the {num} passages (only identifiers) is:
```

3.3 Sliding Window

Due to the token limitations of LLMs, only a limited number of passages can be ranked using the listwise and setwise approaches. To address this limitation, we use a sliding window strategy from [29]. Suppose the first-stage retrieval model returns M passages, we then re-rank these passages in a back-to-first order using the sliding window. In listwise approach, this strategy involves two hyperparameters: window size (w) and step size (s). In setwise approach, we rename window size (w) as num child (c), which are the number of passages at each step of the sorting algorithms.

In listwise, we use the LLM to rank the passages from the last chunk of the list (Figure 4). Then, we slide the window till the beginning of the list in step of (s) and reorder the passage within the window. This process is repeated until the sliding window reaches the first passage in the list.

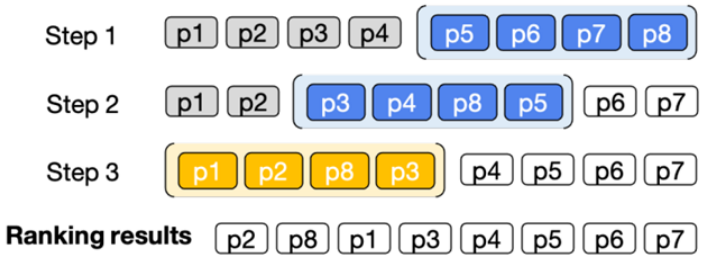

Figure 4: Illustration of listwise sliding window method. The figure rerank 8 passages with a window size w=4 and stepsize s=2. The process starts from the last chunk of the list and slides toward the beginning.

3.4 Sorting Algorithms

To improve the efficiency of the setwise approach, previous studies [33] have introduced sorting algorithms, employing techniques such as *heap sort* and *bubble sort*. These algorithms utilize efficient data structures to selectively compare document pairs, enabling the rapid identification and prioritization of the most relevant documents from the candidate pool. This approach is especially effective for top-k ranking tasks, where only the k most relevant documents need to be ranked. For example, in (Figure 5), 3 passages

are compared at once. Then, the most relevant passage is brought to the top. The sliding window moves backward toward the beginning in steps of $numchild - 1$ until all passages have been rerank. The process will iterate k times to get the top k after reranking step.

Figure 5: Illustration of bubble sort for setwise approach with a num child of 3. Each node represent on passage with the number indicating the level of relevance to the query

Building on the concept of heapsort and bubblesort, we propose tournament sort [15] as a strong candidate due to its competitive performance and resilience to the positional bias of the initial ordering. It organizes elements in a tree-like structure, simulating a tournament. Tournament sorting allows for efficient incremental updates by modifying only the affected parts of the tree. Meanwhile, heapsort would require reconstructing the heap, while bubble sort would need multiple passes to adjust the ranking.

In Figure 6, we demonstrate the application of the tournament sorting algorithm with a *numchild* $c = 4$. This method also incorporates the hyperparameter r, which specifies the number of top passages retrieving in each sliding window to proceed in the next round. In the final round, only top passage is retrieved, representing the final ranking. By applying this parameter, we enable a more comprehensive comparison between sliding windows, mitigating the risk of local maxima. In Figure 6, we set $r = 2$.

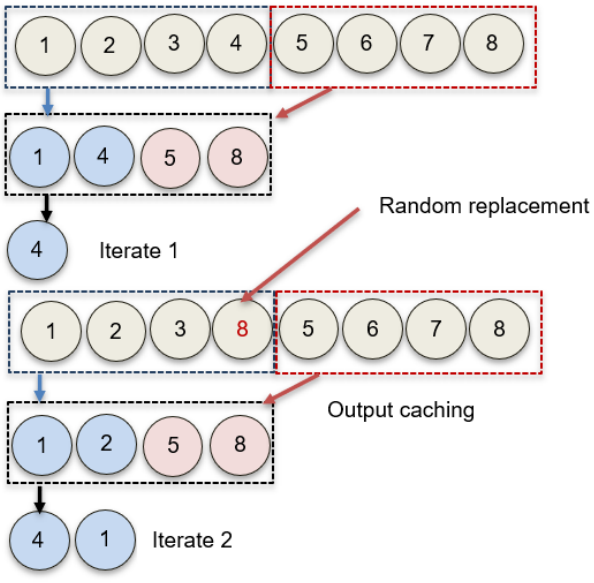

Figure 6: Illustration of tournament sort in setwise prompting on 8 passages with numchild of 4 and top 2 passages retrieving in each sliding window. In first iteration, **P4** is the most relevant to the query and then replaced by a random passage (**P8**) in second iteration.

Also, each sliding window will not be overlapped, allowing matches at the same level to be executed concurrently. This parallelism, combined with output caching, significantly

reduces overall computation time. Meanwhile, this cannot be done in heapsort and bubble sort when the sliding window is overlapping ($step = numchild - 1$).

The process is also iterated until top k passages are retrieved. After each iteration, the top passage is replaced with a randomly selected passage from the list, ensuring the integrity and diversity of the ranking. Future work will investigate how this sampling strategy impacts overall performance.

4 Experiment

4.1 Dataset

The performance of our proposed method is evaluated using three benchmark datasets TREC DL 2019 [19], TREC DL 2020 [18], and TREC COVID [10] from BEIR [31]. These datasets are widely recognized and extensively employed in information retrieval research to evaluate and compare model performance.

TREC is a commonly utilized benchmark dataset in information retrieval research. For our study, we use the test sets from the 2019 and 2020 competitions, which are sample of MSMARCO dataset [22]. The dataset comprises queries sampling from Bing search system. Each question may associate with no answers, one answer, or many answers. The (i) TREC-DL 2019 includes 43 queries, and (ii) TREC-DL 2020 comprises 54 queries.

The BEIR suite comprises a collection of datasets that cover a broad range of retrieval tasks and domains, providing a robust framework for evaluating retrieval models. From BEIR, we selected the TREC-COVID dataset, which is specifically tailored for COVID-19-related document retrieval tasks. TREC-COVID consists of 50 queries and serves as a critical benchmark for testing model performance on domain-specific and real-world retrieval challenges during the pandemic. Each query, on average, is associated with 494 relevant passages.

Dataset	Parent Dataset	No. Queries	Corpus	Relevance Passage/Questions	Relevance Score Range
TREC-COVID	BEIR	50	171K	493.5	0 to 2
TREC DL 19	MSMARCO	43	8.84M	1.1	0 to 3
TREC DL 20	MSMARCO	54	8.84M	1.1	0 to 3

Table 1: Description of each benchmark dataset. The table includes name of dataset, its parent dataset, number of queries, number of passages in corpus, number of relevance passage by corpus, and relevance score range

4.2 Implementation and Artefact Development

For comparison, we evaluate the performance of our proposed method, tournament sort, against SOTA zero-shot setwise prompting methods and competitive listwise approaches. These include the use of heapsort and bubble sort within the setwise approach [37] and instructional listwise prompting with RankGPT [29]. Additionally, we exclusively utilize open-source models that are both popular and frequently recommended in existing literature: FLAN-T5-large [1] and Vicuna-7B [11], to execute the zero-shot ranking task. All LLM-based approaches were utilized to re-rank the top 100 documents initially

retrieved by a BM25 first-stage retrieval system. The initial BM25 rankings for all datasets were produced using the Pyserini Python library [12] with its default configurations.

To align with the prompting token limitation in *listwise*, we set the query length capped at length of 32. We also set the window size (w) to contain 4 documents. The step size (s) is set to 2, and the number of repetitions (r) is set to 5. Furthermore, the combination of a step size of 2 and 5 repetitions is theoretically guaranteed to bring the 10 most relevant documents to the top. For setwise approach, after investigating to find good parameter, we set $numchild$ =3,4 or 5 and $passagelength$ = 100 or 200 for all sorting algorithms depending on dataset and retrieval step and model. As the model produces non-deterministic outputs, each experiment was repeated five times, and the average performance across runs was is to ensure reliable evaluation.

To evaluate the effectiveness of different approaches, we report the performance of NDCG and MRR metrics:

- NDCG (NDCG@10): to evaluate the ranking quality by accounting for the relevance and position of retrieved passages

- MRR (MRR@10): to measure the average rank of the first relevant passage in the retrieved list, highlighting the model's ability to prioritize highly relevant content early in the ranking.

We carried out the experiment on Deakin University Compute Cluster, which is equipped with a dynamic local GPU workstation.

5 Results & Discussion

5.1 Results on Benchmark

Table 2 indicates the ranking performance in the TREC COVID and TREC DL datasets for listwise and setwise reranking method.

All LLM-based zero-shot ranking methods show considerable improvements over the initial BM25 ranking. However, an exception to this trend is observed with the setwise heapsort approach on the TREC COVID and TREC DL 19, and listwise generation on TREC DL 2019 datasets when using the Vicuna1.5-large model. Interestingly, despite its generally underwhelming performance, heapsort achieves the best NDCG@10 score in the TREC COVID dataset on FlanT5 model. This suggests that while the heapsort approach may struggle with robustness across diverse datasets, it has potential in specific contexts.

Retrieval	Model	Rerank	Sorting Algorithms	TREC COVID		TREC DL 2019		TREC DL 2020	
				NDCG@10	MRR@10	NDCG@10	MRR@10	NDCG@10	MRR@10
BM25	None	None	None	0.5947	0.8529	0.5058	0.8529	0.4796	0.8241
	FlanT5 - large	Listwise likelihood [1]		0.7500	0.8820	0.6551	0.9297	0.5993	0.8369
		Listwise generation[1]		0.6925	0.9129	0.5523	0.9099	0.5438	0.9049
		Setwise[1]	Heapsort	**0.7646**	0.9300	0.6577	0.9465	0.6079	0.8824
			Bubblesort	0.7484	0.8973	**0.6848**	**0.9698**	**0.6273**	0.8685
			Tournament sort, $r = 1^*$	0.7454	0.9106	0.6561	0.9426	0.6045	0.8692
			Tournament sort, $r = 2^*$	*0.7532*	*0.94*	*0.6643*	*0.9612*	*0.6183*	0.8892
	Vicuna1.5 - large	Listwise likelihood		0.7034	0.9023	0.5762	0.9194	0.5204	0.8332
		Listwise generation		0.5949	0.8529	0.5058	0.8233	0.5204	0.8332
		Setwise	Heapsort	0.4435	0.87	0.4926	0.9550	0.5204	**0.9141**
			Bubblesort	0.6102	0.8762	0.5731	0.9140	0.5189	0.8489
			Tournament sort, $r = 1^*$	0.7067	**0.955**	0.5538	0.9141	0.5204	**0.9141**
			Tournament sort, $r = 2^*$	0.7210	*0.94*	0.5826	0.9543	0.5666	**0.9141**

Table 2: Comparison of NDCG@10 and MRR@10 scores in TREC COVID, Trec DL 2019, and Trec DL 2020 dataset. Superscript (*) in sorting algorithm indicates proposed method. **Boldface** and *italics* entries indicate the best and the second best performance in specific metrics, respectively.

The bubble sort, on the other hand, achieves the best performance in the FlanT5-large model, particularly in TREC DL 2019 for both metrics and TREC DL 2020 for NDCG@10. However, this trend does not extend to the Vicuna1.5-large model, where its performance remains comparatively weaker. Similar to the heapsort approach, for Vicuna1.5-large, bubble sort exhibits only marginal improvements in NDCG@10 across datasets compared to more competitive methods such as the tournament-based or listwise approaches. These findings highlight a lack of robustness in the bubble sort and heapsort methods.

The proposed tournament sorting method demonstrates the best and competitive

[1]We use source code https://github.com/ielab/llm-rankers with configurations set up as described above

performance across metrics, datasets, and models. This trend is particularly pronounced with the FlanT5-large model using the tournament sort approach with r=2, where it achieves outstanding results in both metrics for TREC-Covid, TREC DL 2019, and NDCG@10 for TREC DL 2020.

Similar to other methods, the tournament approach shows relatively weaker performance with the Vicuna1.5-large model compared to FlanT5-large. However, exceptions are observed for MRR@10 in TREC-Covid and TREC DL 2020, where the method maintains competitive performance. Importantly, the decline in performance with Vicuna1.5-large is significantly less severe than that observed with heapsort and bubble sort. The tournament approach consistently outperforms these methods across all metrics and datasets, particularly when using r=2, as shown in the last row of the table. These findings indicate that the proposed method is more robust in passage reranking tasks, even when applied to less powerful LLMs like Vicuna1.5-large.

5.2 Robustness to Initial Ranking

To evaluate the sensitivity of each setwise sorting algorithm to the initial ranking, we experiment with three variations of BM25 orderings: original BM25, random BM25, and inverse BM25. FlanT5 is selected for this experiment due to its competitive performance on NDCG@10 across different sorting approaches. All configurations remain consistent with the main experiments. The results are presented in Figures 7, 8, and 9 respectively.

It is evident that bubble sort achieves the highest performance with the initial BM25 ranking for both TREC DL 19 and TREC DL 20 datasets. However, this algorithm also exhibits the greatest sensitivity to changes in the initial ranking. Specifically, its performance deteriorates significantly when tested with random BM25 and inverse BM25 orderings, indicating its reliance on the quality of the initial ranking. Furthermore, bubble sort reports the highest standard deviation across the three benchmark datasets, highlighting its inconsistency under different initial retrievals.

In contrast, our proposed methods achieve the best performance, reporting the lowest standard deviation across all datasets, highlighting their robustness to varying initial rankings. In the TREC COVID dataset, tournament sorting not only maintains robustness but also shows improved performance with other BM25 variants (random and inverse orderings). This is in stark contrast to the other algorithms, as both heapsort and bubble sort experience a considerable decline in performance under the same conditions.

For the TREC DL 19 and TREC DL 20 datasets, heapsort also exhibits commendable robustness, delivering competitive results comparable to the proposed method. This

indicates that while heapsort can handle variations in initial rankings relatively well, tournament sorting remains superior in achieving consistent performance across diverse datasets and orderings, enhancing the zero-shot re-ranking ability with LLMs.

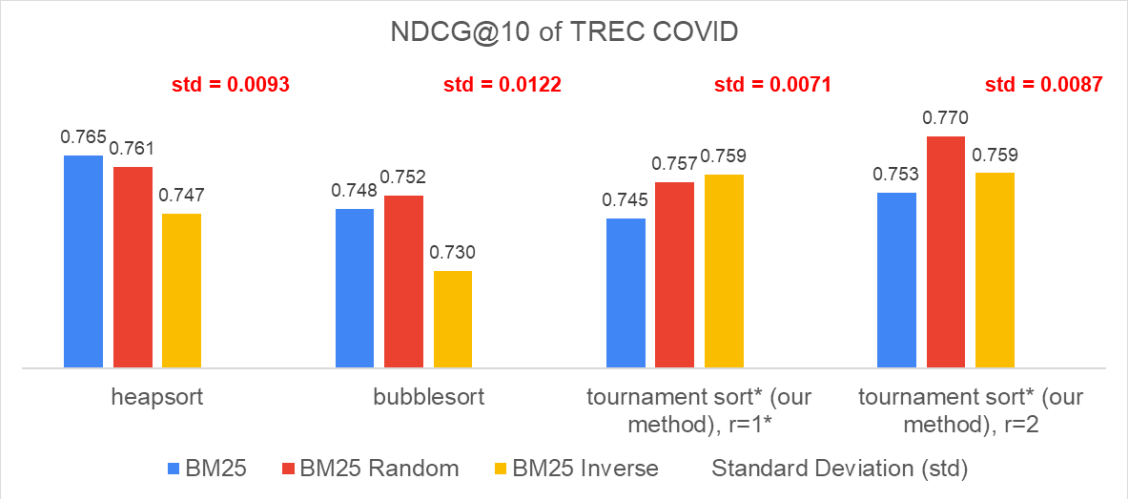

Figure 7: Sensitivity to initial ranking in TREC COVID. Each sorting algorithm is evaluated on different variants of BM25 with a num child value of 3.

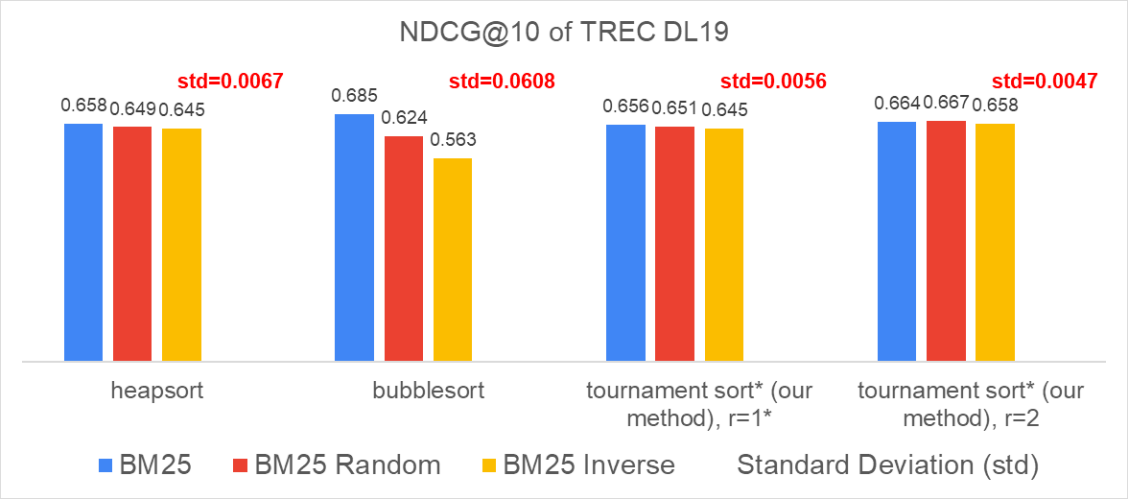

Figure 8: Sensitivity to initial ranking in TREC DL 19. Each sorting algorithm is evaluated on different variants of BM25 with a num child value of 3.

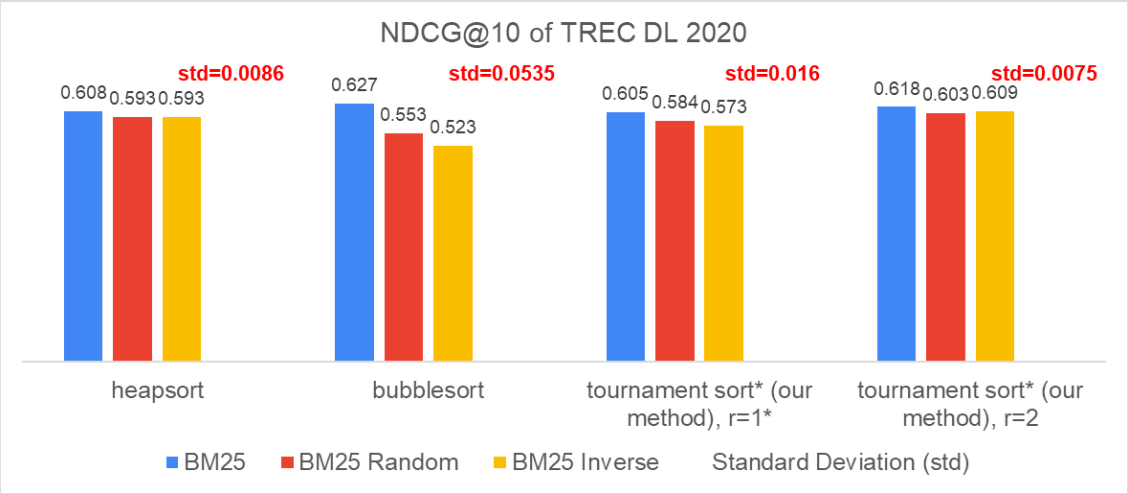

Figure 9: Sensitivity to initial ranking in TREC DL 20. Each sorting algorithm is evaluated on different variants of BM25 with a num child value of 3.

5.3 Effect of Parameter r

To assess the impact of the parameter r, which represents the number of top passages retrieved in each sliding window, on both efficiency and effectiveness, we fixed the value of *num child* at 5 across all benchmark datasets. Subsequently, we evaluated the efficiency and effectiveness of tournament sorting for r values ranging from 1 to 4.

Increasing the value of r results in an exponential rise in running time, with the highest recorded at r=4. This is because the process requires more rounds to identify the top passage in each iteration as multiple passages from each sliding window are carried forward to subsequent rounds. Higher values of r (e.g., 2, 3, 4) typically yield better performance in both NDCG@10 and MRR@10 compared to lower values. However, performance tends to plateau or even decline when r is further increased, highlighting a trade-off between effectiveness and efficiency.

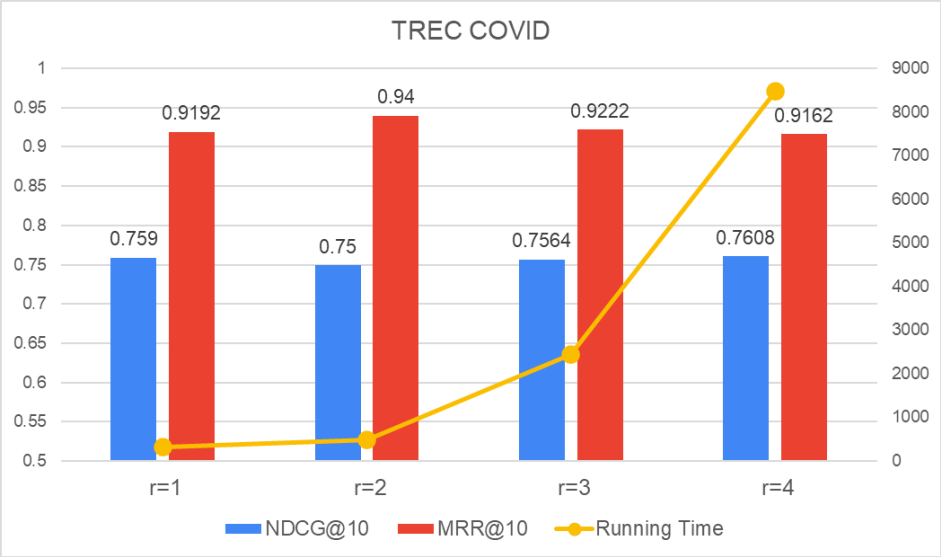

Figure 10: Effect of parameter r on NDCG@10, MRR@10, and Running Time (by seconds) in TREC COVID

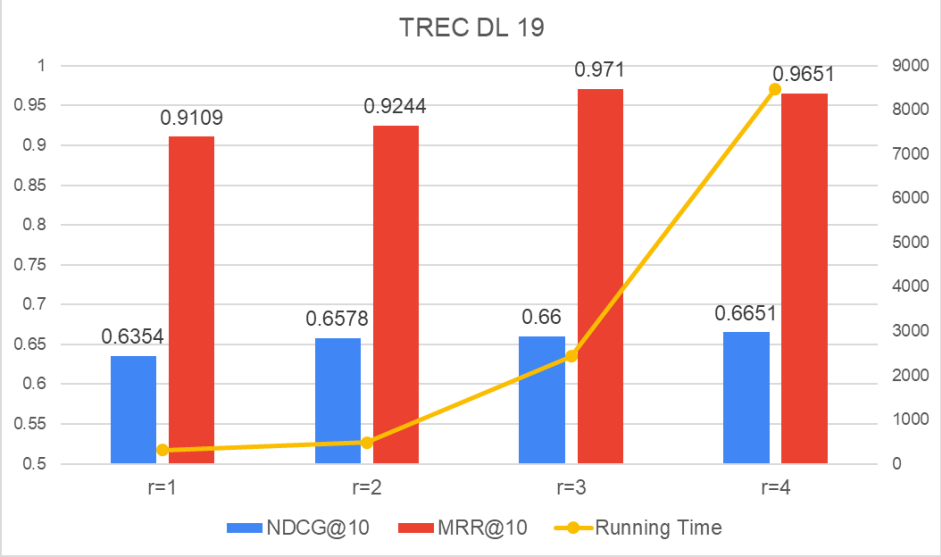

Figure 11: Effect of parameter r on NDCG@10, MRR@10, and Running Time (by seconds) in TREC DL 19

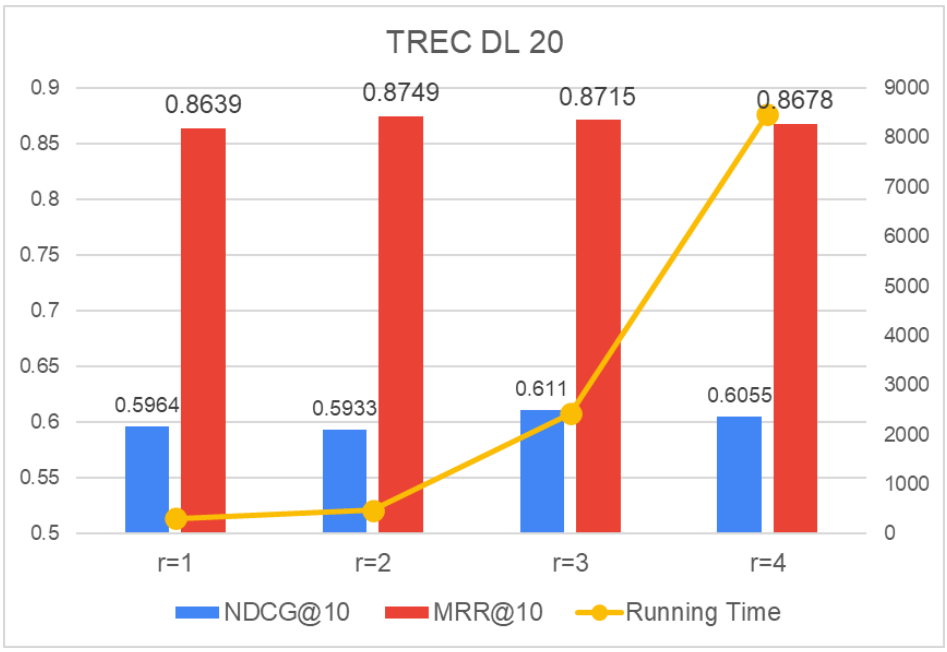

Figure 12: Effect of parameter r on NDCG@10, MRR@10, and Running Time (by seconds) in TREC DL 20

To explore this trade-off, we increased r to 5 and 6, which necessitated setting the sliding window or *num child* parameter to a higher value (c=7). In these cases, the performance in terms of NDCG@10 and MRR@10 was not as strong as for lower values of r. This degradation can be attributed to the token limitations in LLM prompting. To accommodate these constraints, it was necessary to reduce the prompt length for the query and each passage, ultimately compromising effectiveness.

5.4 Comparison to other LLMs

In this section, we evaluate the performance of our proposed method, zero-shot set-wise prompting with tournament sorting, in comparison to a proprietary model (RankGPT 3.5, RankGPT 4) [29] and its fine-tuned counterpart (RankVicuna [27] and RankZephyr [28]).

Category	Model	Method	TREC COVID	TREC DL 19	TREC DL 20
Zeroshot listwise prompting	FlanT5-large (780M)	tournament sort* (our method)	0.7608	0.6643	0.6183
Zeroshot listwise prompting	gpt-3.5-turbo	RANK GPT 3.5 [29]	0.7667	0.658	0.6291
Zeroshot listwise prompting	gpt-4	RANK GPT 4 [29]	0.8551	0.7559	0.7056
GPT 3.5 as teacher model	Vicuna1.5 7B	Rank Vicuna [27]	0.713	0.6682	0.6549
GPT 4 as teacher model	Vicuna1.5 7B	RankZephyr [28]	0.784	0.742	0.7120

Table 3: Comparison of NDCG@10 TREC COVID, Trec DL 2019, and Trec DL 2020 dataset between proposed method and proprietary models.

The table 3 indicates strong performance in fine-tune method. This can be attributed to the fact that both of which are fine-tuned on the MSMARCO dataset, which includes the test sets for DL 19 and DL 20. However, we observe that RankGPT-4 achieves the

highest NDCG@10 across most benchmark datasets, except for TREC DL-20, where RankZephyr records the best performance. Examining tournament sorting, the scores remain competitive in TREC COVID, outperforming RankVicuna and closely on par with RankGPT-3.5. Similarly, in TREC DL 19 and TREC DL 20, the NDCG@10 scores achieved through tournament sorting are comparable to those of RankGPT-3.5 and surpass those of RankVicuna in TREC COVID.

It is also noteworthy that for tournament sorting, we exclusively use the Flan-T5 large model due to resource constraints, which is significantly smaller in size compared to Vicuna1.5 (7B). This indicates that there is potential for further improvement in NDCG@10 if a larger model is utilized or if fine-tuning and hyperparameter optimization are applied more extensively.

Interestingly, despite being fine-tuned, the performance of these models does not consistently surpass their teacher models with zero-shot prompting approaches. For instance, in TREC DL-19, RankZephyr's NDCG@10 is marginally lower than that of RankGPT-4. A similar trend of performance degradation can be observed in TREC COVID for RankVicuna and RankGPT-3.5. These results suggest a lack of generalizability and robustness in current fine-tuning approaches that utilize GPT-based teacher models and training data.

6 Conclusion & Future Work

6.1 Conclusion

In this research, to address key challenges in listwise reranking, we developed a novel tournament sorting algorithm designed to mitigate positional bias and improve the robustness of reranking systems. Through extensive experimentation on benchmark datasets, our approach demonstrated competitive performance, often surpassing existing SOTA zero-shot setwise and listwise methods while maintaining the advantages of open-source accessibility.

Our results indicate that tournament sorting achieves more stable and effective reranking compared to conventional sorting techniques like bubble sort and heap sort. Specifically, our approach showed greater robustness in different initial rankings, reducing the impact of positional bias while maintaining high retrieval performance. Moreover, we demonstrated that increasing the tournament parameter r can enhance the effectiveness of the ranking, although at the cost of increased computational complexity.

A comparative analysis against proprietary models such as RankGPT-3.5 and RankGPT-4 highlights that while closed-source LLMs still maintain an edge in absolute ranking scores, our proposed open-source method remains a viable and scalable alternative. This underscores the potential of open-source LLMs in achieving high-accuracy text retrieval without reliance on expensive and inaccessible proprietary systems.

6.2 Limitation and Future Work

In addition to BM25, we evaluated the tournament sorting method on various retrieval techniques and benchmark datasets to validate its robustness. While positive trends were observed in some cases, such as with the SciFact dataset and the Contriever retrieval method, there were exceptions to this trend. For instance, using SPLADE as the first-step ranking method did not yield comparable performance. Detailed performance results are provided in the Appendix.

The table 7 shows detailed performance of NDCG and MRR on TREC DL 19 using SPLADE-ED as retrieval step. Interestingly, almost all of setwise and listwise performance have decreased compared to original ranking. Hence, there is a need for further exploration to investigate this trend and improve the robustness of the method.

Another potential area for improvement lies in the models used during this phase of testing. Currently, we have only evaluated the large versions of open-source models, such

as Flan-T5 and Vicuna1.5. To enhance the generalizability of the tournament sorting method, future work will involve exploring a broader range of models, particularly those with a larger number of parameters. This could provide valuable insights into how model size and architecture impact the performance and robustness of tournament sorting, potentially unlocking further improvements in effectiveness and efficiency.

Retrieval	**Model**	**Rerank**	**Sorting Algorithms**	**TREC DL 2019**	
				NDCG@10	MRR@10
SPLADE-ED	None	None	None	0.7308	0.9729
	FlanT5 - large	Listwise likelihood		0.6594	0.8617
		Setwise	Heapsort	0.6361	0.9147
			Bubblesort	0.7286	0.9597
			Tournament sort, $r = 1^*$	0.6231	0.8527
	Vicuna1.5 - large	Listwise likelihood		0.7366	0.9535
		Setwise	Heapsort	0.5539	0.9801
			Bubblesort	0.7361	0.9729
			Tournament sort, $r = 1^*$	0.5979	0.9419

Table 4: Comparison of NDCG@10 and MRR@10 in Trec DL 2019 with SPLADE-ED[2].

[2]We use source code https://github.com/ielab/llm-rankers for other reranking method

References

[1] R. Colin, S. Noam, R. Adam, L. Katherine, N. Sharan, M. Michael, Z. Yanqi, L. Wei, and J. L. Peter, *Exploring the limits of transfer learning with a unified text-to-text transformer*, (2023).

[2] S. Devendra, L. Mike, J. Mandar, A. Armen, Y. Wen-tau, P. Joelle, and Z. Luke, *Improving passage retrieval with zero-shot question generation*, (2022).

[3] A. Drozdov, H. Zhuang, Z. Dai, Z. Qin, R. Rahimi, X. Wang, D. Alon, M. Iyyer, A. McCallum, D. Metzler, and K. Hui, *Parade: Passage ranking using demonstrations with llms*, Association for Computational Linguistics, 2023.

[4] T. Formal, B. Piwowarski, and S. Clinchant, *Splade: Sparse lexical and expansion model for first stage ranking*, ACM, 2021.

[5] R. Gangi Reddy, J. Doo, Y. Xu, M. A. Sultan, D. Swain, A. Sil, and H. Ji, *First: Faster improved listwise reranking with single token decoding*, Association for Computational Linguistics, 2024, pp. 8642–8652.

[6] L. Gao, Z. Dai, and J. Callan, *Coil: Revisit exact lexical match in information retrieval with contextualized inverted list*, Association for Computational Linguistics.

[7] L. Gao, X. Ma, J. Lin, and J. Callan, *Precise zero-shot dense retrieval without relevance labels*, Association for Computational Linguistics, 2022.

[8] I. Gautier, C. Mathilde, H. Lucas, R. Sebastian, B. Piotr, J. Armand, and G. Edouard, *Unsupervised dense information retrieval with contrastive learning*, (2021).

[9] O. Khattab and M. Zaharia, *Colbert: Efficient and effective passage search viacontextualized late interaction over bert*, ACM, pp. 39–48.

[10] R. Kirk, A. Tasmeer, B. Steven, D.-F. Dina, L. Kyle, S. Ian, V. Ellen, L. W. Lucy, and R. H. William, *Searching for scientific evidence in a pandemic: An overview of trec-covid*, (2021).

[11] Z. Lianmin, C. Wei-Lin, S. Ying, Z. Siyuan, W. Zhanghao, Z. Yonghao, L. Zi, L. Zhuohan, L. Dacheng, P. X. Eric, Z. Hao, E. G. Joseph, and S. Ion, *Judging llm-as-a-judge with mt-bench and chatbot arena*, (2023).

[12] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira, *Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations*, ACM, pp. 2356–2362.

[13] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, *Lost in the middle: How language models use long contexts*, Transactions of the Association for Computational Linguistics, 12 (2024), pp. 157–173.

[14] B. Luiz, A. Hugo, F. Marzieh, and N. Rodrigo, *Inpars: Data augmentation for information retrieval using large language models*, SIGIR '22: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, (2022), pp. 2387 – 2392.

[15] K. McLuckie and A. Barber, *Tournament Sort*, Macmillan Education UK, London, 1986, pp. 68–86.

[16] A. Mikhailiuk, C. Wilmot, M. Perez-Ortiz, D. Yue, and R. K. Mantiuk, *Active sampling for pairwise comparisons via approximate message passing and information gain maximization*, IEEE, 2020, pp. 2559–2566.

[17] D. Nagumothu, B. Ofoghi, and P. W. Eklund, *Semantic Triple-Assisted Learning for Question Answering Passage Re-ranking*, Springer Nature Switzerland, 2023, pp. 249–264.

[18] C. Nick, M. Bhaskar, Y. Emine, and C. Daniel, *Overview of the trec 2020 deep learning track*, (2021).

[19] C. Nick, M. Bhaskar, Y. Emine, C. Daniel, and M. V. Ellen, *Unknown article*, Overview of the TREC 2019 deep learning track, (2020).

[20] R. Nogueira, Z. Jiang, R. Pradeep, and J. Lin, *Document ranking with a pretrained sequence-to-sequence model*, Association for Computational Linguistics, 2022.

[21] OpenAI, *Gpt-4 technical report*, (2024).

[22] B. Payal, C. Daniel, C. Nick, D. Li, G. Jianfeng, L. Xiaodong, M. Rangan, M. Andrew, M. Bhaskar, N. Tri, R. Mir, S. Xia, S. Alina, T. Saurabh, and W. Tong, *Ms marco: A human generated machine reading comprehension dataset*, (2016).

[23] L. Qi, W. Bo, W. Nan, and M. Jiaxin, *Leveraging passage embeddings for efficient listwise reranking with large language models*, (2024).

[24] Z. Qin, R. Jagerman, K. Hui, H. Zhuang, J. Wu, L. Yan, J. Shen, T. Liu, J. Liu, D. Metzler, X. Wang, and M. Bendersky, *Large language models are effective text rankers with pairwise ranking prompting*, Association for Computational Linguistics, 2023.

[25] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: Bm25 and beyond*, Foundations and Trends® in Information Retrieval, 3 (2009), pp. 333–389.

[26] N. Rodrigo and C. Kyunghyun, *Passage re-ranking with bert*, (2019).

[27] P. Ronak, S. Sahel, and L. Jimmy, *Rankvicuna: Zero-shot listwise document reranking with open-source large language models*, (2023).

[28] P. Ronakm, S. Sahel, and L. Jimmy, *Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze!*, (2023).

[29] W. Sun, L. Yan, X. Ma, S. Wang, P. Ren, Z. Chen, D. Yin, and Z. Ren, *Is chatgpt good at search? investigating large language models as re-ranking agents*, Association for Computational Linguistics, 2023.

[30] R. Tang, C. Zhang, X. Ma, J. Lin, and F. Ture, *Found in the middle: Permutation self-consistency improves listwise ranking in large language models*, Association for Computational Linguistics, pp. 2327–2340.

[31] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, *BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models*, in Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021.

[32] Z. Xinyu, H. Sebastian, L. Patrick, T. Raphael, and L. Jimmy, *Rank-without-gpt: Building gpt-independent listwise rerankers on open-source large language models*, (2023).

[33] M. Xueguang, Z. Xinyu, P. Ronak, and L. Jimmy, *Zero-shot listwise document reranking with a large language model*, (2023).

[34] S. Yoon, E. Choi, J. Kim, H. Yun, Y. Kim, and S.-W. Hwang, *Listt5: Listwise reranking with fusion-in-decoder improves zero-shot retrieval*, Association for Computational Linguistics, 2024, pp. 2287–2308.

[35] H. Zhuang, Z. Qin, K. Hui, J. Wu, L. Yan, X. Wang, and M. Bendersky, *Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels*, Association for Computational Linguistics, 2024.

[36] H. Zhuang, Z. Qin, R. Jagerman, K. Hui, J. Ma, J. Lu, J. Ni, X. Wang, and M. Bendersky, *Rankt5: Fine-tuning t5 for text ranking with ranking losses*, ACM, 2023.

[37] S. Zhuang, H. Zhuang, B. Koopman, and G. Zuccon, *A setwise approach for effective and highly efficient zero-shot ranking with large language models*, ACM, pp. 38–47.

A Appendix

A.1 Other retrieval results

Retrieval	Model	Rerank	Sorting Algorithms	TREC COVID	
				NDCG@10	MRR@10
Contriever	None	None	None	0.5694	0.8047
	FlanT5 - large	Listwise likelihood		0.6676	0.8513
		Setwise	Heapsort	0.6818	0.8957
			Bubblesort	0.6959	0.9017
			Tournament sort, $r = 1^*$	0.6667	0.8857
			Tournament sort, $r = 2^*$	0.6818	0.859
	Vicuna1.5 - large	Listwise likelihood		0.6439	0.9033
		Setwise	Heapsort	0.5450	0.8246
			Bubblesort	0.6373	0.869
			Tournament sort, $r = 1^*$	0.6131	0.889
			Tournament sort, $r = 2^*$	0.6546	0.8867

Table 5: Comparison of NDCG@10 and MRR@10 scores in TREC COVID with Contriever retrieval step. Superscript (*) in sorting algorithm indicates proposed method.

Retrieval	Model	Rerank	Sorting Algorithms	TREC COVID	
				NDCG@10	MRR@10
SPLADE-ED	None	None	None	0.7274	0.9117
	FlanT5 - large	Listwise likelihood		0.7439	0.88
		Setwise	Heapsort	0.7767	0.824
			Bubblesort	0.7782	0.9667
			Tournament sort, $r = 1^*$	0.7223	0.9167
			Tournament sort, $r = 2^*$	0.7348	0.8667
	Vicuna1.5 - large	Listwise likelihood		0.7724	0.94
		Setwise	Heapsort	0.5945	0.9169
			Bubblesort	0.7589	0.9467
			Tournament sort, $r = 1^*$	0.7237	0.945
			Tournament sort, $r = 2^*$	0.742	0.9133

Table 6: Comparison of NDCG@10 and MRR@10 scores in TREC COVID with SPLADE-ED retrieval step. Superscript (*) in sorting algorithm indicates proposed method.

A.2 Benchmark on Scifact Dataset

Retrieval	Model	Rerank	Sorting Algorithms	TREC COVID	
				NDCG@10	MRR@10
BM25	None	None	None	0.6789	0.6457
	Vicuna1.5-large	Listwise likelihood		0.6756	0.6292
		Setwise	Heapsort	0.6432	0.6217
			Bubblesort	0.665	0.6170
			Tournament sort, $r = 1^*$	0.6850	0.6250
			Tournament sort, $r = 2^*$	0.7034	0.6672

Table 7: Comparison of NDCG@10 and MRR@10 scores on Scifact with SPLADE-ED retrieval step. Superscript (*) in sorting algorithm indicates proposed method.