# A Setwise Approach for Effective and Highly Efficient Zero-shot Ranking with Large Language Models


Shengyao Zhuang*
CSIRO
Brisbane, Australia
shengyao.zhuang@csiro.au

Honglei Zhuang
Google Research
Mountain View, USA
hlz@google.com

Bevan Koopman
CSIRO
Brisbane, Australia
bevan.koopman@csiro.au

Guido Zuccon
The University of Queensland
St Lucia, Australia
g.zuccon@uq.edu.au



## ABSTRACT

We propose a novel zero-shot document ranking approach based on Large Language Models (LLMs): the Setwise prompting approach. Our approach complements existing prompting approaches for LLM-based zero-shot ranking: Pointwise, Pairwise, and Listwise. Through the first-of-its-kind comparative evaluation within a consistent experimental framework and considering factors like model size, token consumption, latency, among others, we show that existing approaches are inherently characterised by trade-offs between effectiveness and efficiency. We find that while Pointwise approaches score high on efficiency, they suffer from poor effectiveness. Conversely, Pairwise approaches demonstrate superior effectiveness but incur high computational overhead. Our Setwise approach, instead, reduces the number of LLM inferences and the amount of prompt token consumption during the ranking procedure, compared to previous methods. This significantly improves the efficiency of LLM-based zero-shot ranking, while also retaining high zero-shot ranking effectiveness. We make our code and results publicly available at https://github.com/ielab/llm-rankers.



## CCS CONCEPTS

• **Information systems** → **Language models**.

## KEYWORDS

Large Language Model for Zero-shot ranking, setwise prompting, sorting algorithm

**ACM Reference Format:**
Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2024. A Setwise Approach for Effective and Highly Efficient Zero-shot Ranking with Large Language Models. 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, 10 pages. https://doi.org/10.1145/3626772.3657813


## 1 INTRODUCTION

Large Language Models (LLMs) such as GPT-3 [3], FlanT5 [33], Gemini [24], LLaMa2 [27], and PaLM [4] are highly effective across a diverse range of natural language processing tasks under the zero-shot settings [1, 3, 10, 32]. Notably, these LLMs have also been adapted for zero-shot document ranking. [12, 15, 20–23]. Using LLMs in zero-shot ranking tasks can be broadly categorized into three main approaches: *Pointwise* [12, 22, 37], *Listwise* [15, 20, 23], and *Pairwise* [21]. These approaches employ different prompting strategies to instruct the LLM to output a relevance estimation for each candidate document and rank documents accordingly. Compared to traditional neural ranking methods [14], these LLM-based rankers do not require any further supervised fine-tuning and exhibit strong zero-shot ranking capabilities. While these LLM-based zero-shot ranking approaches have been successful individually, it is worth noting that there has been a lack of fair comparison in the literature regarding their effectiveness, and in particular, their efficiency within the exact same experimental framework. This includes factors such as utilizing the same size LLM, evaluation benchmarks, and computational resources. We believe it is critical to establish a rigorous framework for evaluating these LLM-based zero-shot ranking approaches. By doing so, we can draw meaningful conclusions about their comparative effectiveness and efficiency.

Thus, in this paper, we first conduct a systematic evaluation of all existing approaches within a consistent experimental environment. In addition to assessing ranking effectiveness, we also compare the efficiency of these methods in terms of computational expense and query latency. Our findings indicate that the *Pairwise* approach emerges as the most effective but falls short in terms of efficiency even with the assistance of sorting algorithms aimed at improving efficiency. Conversely, the *Pointwise* approach stands out as the most efficient but lags behind other methods in terms of ranking effectiveness. The *Listwise* approach, which relies solely on the generation of document labels in order, can strike a middle ground between efficiency and effectiveness but this varies considerably based on configuration, implementation and evaluation dataset (highlighting the importance of thoroughly evaluating these model under multiple settings). Overall, these comprehensive results offer an understanding of the strengths and weaknesses of LLM-based zero-shot ranking approaches, providing valuable insights for those seeking to select the most suitable approach for real-world applications.

Having considered all the different approaches and their results in terms of efficiency and effectiveness tradeoffs, we set about devising a method that was both effective and efficient. Our approach was to take the most effective model (*Pairwise*) and to enhance its efficiency (without seriously compromising effectiveness). Our solution is a novel *Setwise* prompting approach. This concept stems from our realisation that the sorting algorithms employed by *Pairwise* approaches can be accelerated by comparing multiple documents, as opposed to just a pair at a time.


*Corresponding author.





*SIGIR '24, July 14–18, 2024, Washington, DC, USA*



ACM ISBN 979-8-4007-0431-4/24/07.
https://doi.org/10.1145/3626772.3657813

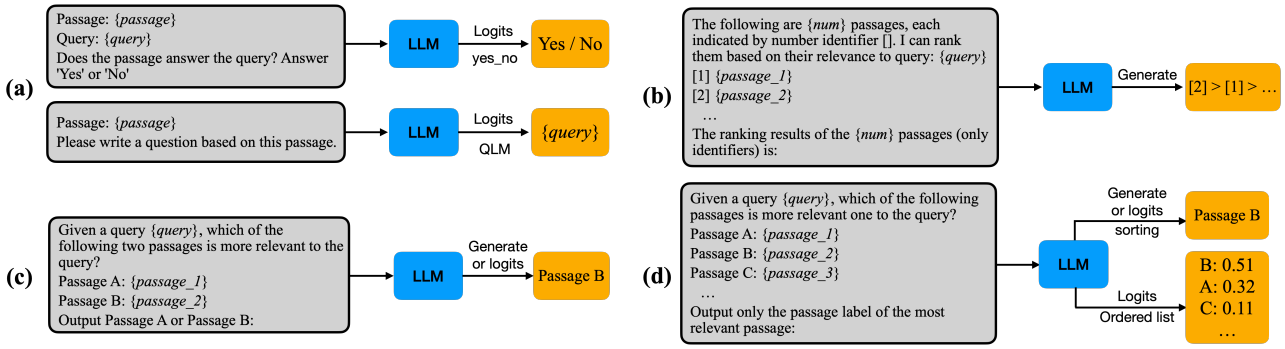



**Figure 1: Different prompting strategies. (a) *Pointwise*, (b) *Listwise*, (c) *Pairwise* and (d) our proposed *Setwise*.**

Our *Setwise* prompting approach instructs LLMs to select the most relevant document to the query from a set of candidate documents. This straightforward adjustment allows the sorting algorithms to infer relevance preferences for more than two candidate documents at each step, thus significantly reducing the total number of comparisons required; this leads to substantial savings in computational resources. Furthermore, beyond the adjustment to *Pairwise* approaches, *Setwise* prompting allows for the utilization of model output logits to estimate the likelihood of ranks of document labels, a capability not feasible in existing *Listwise* approaches, which solely rely on document label ranking generation —– a process that is slow and less effective. We *Setwise* and other existing approaches under the same experimental settings to provide a clear and consistent comparison. Our results show that the incorporation of our *Setwise* prompting substantially improves the efficiency of both *Pairwise* and *Listwise* approaches. In addition, Setwise sorting enhances *Pairwise* and *Listwise* robustness to variations in the internal ordering quality of the initial rankings: no matter what the initial ordering of the top-k documents to rank is, our method provides consistent and effective results. This is unlike other methods that are highly susceptible to such initial ordering.

To conclude, this paper makes three key contributions to our understanding of LLM-based zero-shot ranking approaches:

(1) We introduce an innovative *Setwise* prompting approach that enhances the sorting algorithms employed in the *Pairwise* method, resulting in highly efficient zero-shot ranking with LLMs.
(2) We conduct a systematic examination of all existing LLM-based zero-shot ranking approaches and our novel *Setwise* approach under strict and consistent experimental conditions, including efficiency comparisons which have been overlooked in the literature. Our comprehensive empirical evaluation on popular zero-shot document ranking benchmarks offers valuable insights for practitioners.
(3) We further adapt how our *Setwise* prompting approach computes rankings to the *Listwise* approach, leveraging the model output logits to estimate the likelihood of rankings. This leads to a more effective and efficient *Listwise* zero-shot ranking.

## 2 BACKGROUND & RELATED WORK

There are three main prompting approaches for zero-shot document ranking employing LLMs: *Pointwise* [12, 22], *Listwise* [15, 20, 23], and *Pairwise* [21]. In this section, we delve into the specifics of these while situating our work within the existing literature. As a visual aid we will refer to Figure 1 as we discuss each method.

### 2.1 *Pointwise* prompting approaches

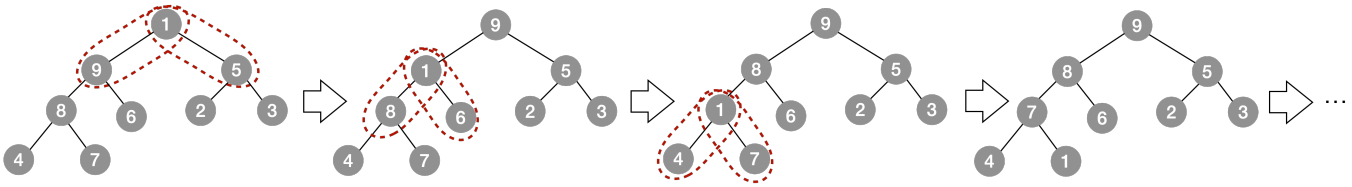
Figure 1a shows pointwise approaches. There are two popular directions of prompting LLMs for ranking documents in a pointwise manner: *generation* and *likelihood*. In the generation approach, a "yes/no" generation technique is used: LLMs are prompted to generate whether the provided candidate document is relevant to the query, with the process repeated for each candidate document. Subsequently, these candidate documents are re-ranked based on the normalized likelihood of generating a "yes" response [12, 17]. The likelihood approach involves query likelihood modelling (QLM) [18, 36, 39], wherein LLMs are prompted to produce a relevant query for each candidate document. The documents are then re-ranked based on the likelihood of generating the actual query [22]. It is worth noting that both pointwise methods require access to the output logits of the model to be able to compute the likelihood scores. Thus, it is not possible to use closed-sourced LLMs to implement these approaches if the corresponding APIs do not expose the logits values: this is the case for example of GPT-4.

### 2.2 *Listwise* prompting approaches

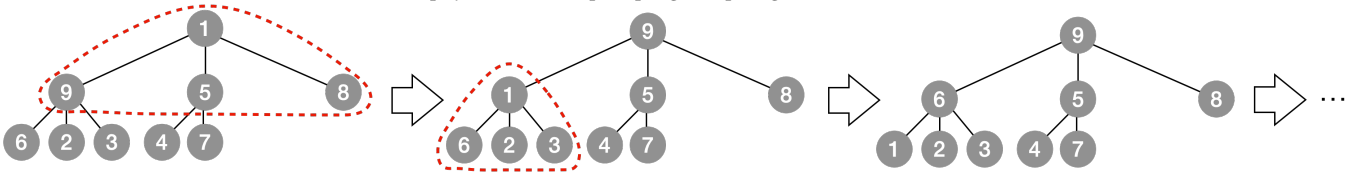
Figure 1b shows listwise approaches. Here the LLMs receive a query along with a list of candidate documents and are prompted to generate a ranked list of document labels based on their relevance to the query [15, 20, 23]. However, due to the limited input length allowed by LLMs, including all candidate documents in the prompt is not feasible. To address this, current listwise approaches use a sliding window method. This involves re-ranking a window of candidate documents, starting from the bottom of the original ranking list and progressing upwards. This process can be repeated multiple times to achieve an improved final ranking and allows for early stopping mechanisms to target only the top-$k$ ranking, thereby conserving computational resources. In contrast to pointwise methods, which utilize the likelihood value of the output tokens for ranking documents, listwise approaches rely on the more efficient process of generation of the ranking list.

### 2.3 *Pairwise* prompting approaches

Figure 1c shows pairwise approaches. LLMs are prompted with a query alongside a pair of documents, and are asked to generate the label indicating which document is more relevant to the



(a) Heapify with Pairwise prompting (comparing 2 documents at a time).



(b) Heapify with our Setwise prompting (comparing 4 documents at a time).

(c) Bubble sort with Pairwise prompting (comparing 2 documents at a time).

(d) Bubble sort with our Setwise prompting (comparing 3 documents at a time).

**Figure 2: Illustration of the impact of Setwise Prompting vs. Pairwise Prompting on Sorting Algorithms. Nodes are documents, numbers in nodes represent the level of relevance assigned by the LLM (higher is more relevant).**

query [19, 21]. To re-rank all candidate documents, a basic method, called *AllPairs*, involves generating all possible permutations of document pairs from the candidate set. Pairs are independently then fed into the LLM, and the preferred document for each pair is determined. Subsequently, an aggregation function is employed to assign a score to each document based on the inferred pairwise preferences, and the final ranking is established based on the total score assigned to each document [19]. However, this aggregation-based approach suffers from high query latency: LLM inference on all document pairs can be computationally expensive. To address this efficiency issue in pairwise approaches, prior studies have introduced sampling [8, 16] and sorting [21] algorithms. In this paper, we focus on sorting algorithms because, assuming an LLM can provide ideal pairwise preferences, the sorting algorithms offer the theoretical assurance of identifying the top-$k$ most relevant documents from the candidate pool. In prior work [21], two sorting algorithms [9], *heap sort* and *bubble sort*, were employed. Unlike *AllPairs*, these algorithms leverage efficient data structures to selectively compare document pairs, which can quickly pull the most relevant documents out from the candidate pool and place them at the top of the final ranking. This is particularly suitable for the top-$k$ ranking task, where only a ranking of the $k$ most relevant documents is needed. These sorting algorithms provide a stopping mechanism that prevents the need to rank all candidate documents.

From a theoretical standpoint the differences and relative advantages among these three families of zero-shot document ranking that employ LLMs are clear. However, from an empirical standpoint there has been no fair and comprehensive evaluation of these techniques in terms of effectiveness vs. efficiency, and across factors such as sizes of LLMs, benchmarks, and computational resources.

## 2.4 Other Directions in using LLMs for Ranking

The three families of approaches outlined above directly use prompting of LLMs to infer document relevance to a given query, leveraging LLMs for document re-ranking in a zero-shot setting with no training data required for the ranking task. An alternative direction in using LLMs for retrieval has also emerged, where LLMs are instead used as text embedding models for dense document retrieval [2, 11, 30, 31]. Because these methods create document representations independently of query representations, they can be used as a first-stage document retriever (i.e. in all effects as a bi-encoder architecture), rather than being limited to be used as a re-ranker as instead are the approaches from the three families we have reviewed above. However, these methods require performing contrastive learning training to enable the generative LLM to act as a text-embedding model. An exception to this is the recent PromptReps method [38], which does not require contrastive training, achieving document (and query) embedding simply via prompt engineering. Another feature of PromptReps is the ability to obtain at the same time both a dense and a sparse representation of documents and queries.

# 3 SETWISE RANKING PROMPTING

In this section, we discuss the limitations present in the current LLM-based zero-shot ranking methods. We then describe our proposed *Setwise* approach and how it addresses these limitations.

## 3.1 Limitations of Current Approaches

The efficiency of LLM-based zero-shot ranking methods hinges on two critical dimensions.

**Table 1: Properties of different methods. Logits: requires access to the LLM logits. Generate: requires actually generating tokens. Batching: allows for batch inference. Top-$k$: allows for early stopping once top-$k$ most relevant documents found. # LLM calls: the number of LLM forward passes needed in the worst case. ($N$: number of documents to re-rank. $r$: number of repeats. $s$: step size for sliding window. $k$: number of top-$k$ documents to find. $c$: number of compared documents at each step.)**

| Methods | Logits | Generate | Batching | Top-$k$ | # LLM calls |
|---|---|---|---|---|---|
| *pointwise.qlm* | ✓ | | ✓ | | $O(N)$ |
| *pointwise.yes_no* | ✓ | | ✓ | | $O(N)$ |
| *listwise.generation* | | ✓ | | ✓ | $O(r * (N/s))$ |
| *listwise.likelihood* | ✓ | | | ✓ | $O(r * (N/s))$ |
| *pairwise.allpair* | ✓ | ✓ | ✓ | | $O(N^2 - N)$ |
| *pairwise.heapsort* | ✓ | ✓ | | ✓ | $O(k * \log_2 N)$ |
| *pairwise.bubblesort* | ✓ | ✓ | | ✓ | $O(k * N)$ |
| *setwise.heapsort* | ✓ | ✓ | | ✓ | $O(k * \log_c N)$ |
| *setwise.bubblesort* | ✓ | ✓ | | ✓ | $O(k * (N/(c-1)))$ |

First, the number of LLM inferences significantly impacts efficiency. Given that LLMs are large neural networks with billions of parameters, inference is computationally intensive. Hence, an increased number of LLM inferences introduces a considerable computational overhead. This is notably observed in the current *Pairwise* approach, which is inefficient due to the extensive need for inferring preferences for the many document pairs. While sorting algorithms offer some relief, they do not entirely mitigate the efficiency issue.

Second, the number of LLM-generated tokens per inference plays a pivotal role. LLMs employ a transformer decoder for autoregressive token generation, where the next token generation depend on previously tokens generated. Each additional generated token requires an extra LLM inference. This accounts for the inefficiency of the existing *Listwise* approach, which relies on generating an entire ranking of document label lists, often requiring a substantial number of generated tokens.

Next, we introduce our new *Setwise* prompting approaches, designed to overcome the efficiency limitations of both *Pairwise* and *Listwise* methods by minimizing the number of required LLM inferences and leveraging the logits produced by the LLM.

## 3.2 Speeding-up Pairwise with Setwise

To solve the inefficiency issue of these approaches, we propose a novel *Setwise* prompting approach. Our prompt, as illustrated in Figure 1d, instructs the LLM to select the most relevant document for the given query from a set of documents, hence the term *Setwise* prompting. We specifically treat the collection of documents as an unordered set and later experiments will show that *Setwise* prompting is quite robust to document ordering.

With our prompt, sorting-based *Pairwise* approaches can be considerably accelerated. This is because the original *heap sort* and *bubble sort* algorithm used in the *Pairwise* approach only compares a pair of documents at each step in the sorting process, as illustrated in Figure 2a and 2c. These sorting algorithms can be sped up by comparing more than two documents at each step. For example, in the *heap sort* algorithm, the "heapify" function needs to be invoked for each subtree, where the parent node must be swapped with the child node with the highest value if it exceeds the parent value. In the case of Figure 2a, to perform "heapify" with pairwise prompting, a minimum of 6 comparisons (each root node paired with each child node) are required. Conversely, if we increase the number of child nodes in each subtree to 3 and can compare 4 nodes at a time, only 2 comparisons are needed to "heapify" a tree with 9 nodes, as illustrated in Figure 2b. Similarly, for the *bubble sort* algorithm, if we can compare more than a pair of documents at a time, each "bubbling" process will be accelerated. For instance, in Figure 2c, there are 4 comparisons in total, but in Figure 2d, with the ability to compare 3 documents at once, only 2 comparisons are required to be able to bring the node with the largest value to the top. Our *Setwise* prompting is designed to instruct LLMs to compare the relevance of multiple documents at a time, making it well-suited for this purpose.

## 3.3 Listwise Likelihoods with Setwise

Our *Setwise* prompting can also accelerate the ranking process for the *Listwise* approach. The original *Listwise* method relies on the LLM's next token generation to produce the complete ordered list of document labels at each step of the sliding window process, as illustrated in Figure 1b. As we discussed, generating the document label list is computationally intensive, because the LLM must do one inference for each next token prediction. On the other hand, the LLM may generate results in an unexpected format or even decline to generate the desired document label list [23], thus harming effectiveness. Fortunately, if we have access to the LLM's output logits, these issues can be avoided by evaluating the likelihood of generating every conceivable document label list and then selecting the most probable one as the output. Regrettably, this is only theoretically possible, but in practice, it is unfeasible for the existing *Listwise* approach due to the very large number of possible document label permutation, which implies that the process of likelihood checking may actually become even more time-consuming than generating the list itself.

*Setwise* prompting again provides a solution: we can easily derive an ordered list of document labels from the LLM output logits. This is done by assessing the likelihood of each document label being chosen as the most relevant, as shown in Figure 1d. This straightforward trick markedly accelerates *Listwise* ranking, as it requires only a single forward pass of the LLM, and also guarantees that the output matches the desired document label list.

## 3.4 Advantages of Setwise

We summarize and compare the key different properties of existing zero-shot LLM ranking approaches along with our proposed *Setwise* prompting approach in Table 1. Notably, *pointwise.qlm*, *pointwise.yes_no* and *pairwise.allpair* require a brute-force of LLM inference for all available documents relevance or preferences. Thus, they are unable to facilitate early-stopping for the top-$k$ ranking. However, these approaches do allow batch inferences, hence the maximum GPU memory utilization could be easily achieved by using the highest batch size. On the other hand, other approaches use sorting algorithms, enabling early-stopping once the top-$k$ most relevant documents are identified. However, this compromises the feasibility of batching inference, as the LLM inference at each step of the sorting algorithms relies on the results from the preceding step. Our *Setwise* prompting empowers the previous *Listwise* approach (*listwise.generation*), which relied on LLM's next token

generations, to now utilize the LLM's output logits. We refer to the *Listwise* approach that incorporates our *Setwise* prompt as *listwise.likelihood*. Finally, comparing with *Pairwise* approaches, our *Setwise* prompting has fewer LLM calls by comparing a minimum of $c \geq 3$ documents at each step of the sorting algorithms.

On the other hand, model output calibration might be a concern for *Pointwise* methods because each document's relevance is inferred independently. Consequently, ranking documents based on *Pointwise* relevance scores necessitates calibration [21]. However, for our *Setwise* method (as well as for *Pairwise* and *Listwise*) the model output logits are derived from the input of multiple documents, which do not directly represent the relevance of a single document but rather serve as an indicator of preference among documents. Thus, calibration is not necessary.

# 4 EXPERIMENTS

## 4.1 Datasets and evaluations

The first objective of this study is to contribute a fair and comprehensive evaluation of existing LLM-based zero-shot ranking methods in terms ranking effectiveness and efficiency. To achieve this goal, we carried out extensive empirical evaluations using well-established document ranking datasets: the TREC Deep Learning 2019 [6] and 2020 [5], along with the BEIR benchmark datasets [25]. To guarantee a fair comparison across different approaches, we tested all of the methods using the same open-source Flan-t5 LLMs [33], available on the Huggingface model hub in various sizes (780M, 3B, and 11B parameters). All LLM methods were used to re-rank 100 documents retrieved by a BM25 first-stage retriever. In order to optimize efficiency, the focus was on a top-$k$ ranking task, whereby the re-ranking process stopped as soon as the top-$k$ most relevant documents were identified and ranked. Here, we set $k = 10$. The effectiveness of different approaches was evaluated using the NDCG@10 metric, which serves as the official evaluation metric for the employed datasets.

Efficiency was evaluated with the following metrics:

- *The average number of LLM inferences per query.* LLMs have limited input length. Thus, to re-rank 100 documents, multiple LLM inferences are often needed. It's important to note that an increased number of LLM inferences translates to higher computational demands. Thus, we regard this as an efficiency metric worth considering.
- *The average number of prompt tokens inputted to the LLMs per query.* This metric takes into account the actual average quantity of input tokens required in the prompts for each method to re-rank 100 documents per query. Given that self-attention mechanisms in transformer-based LLMs become prohibitively costly for a large number of input tokens [29], an increase in tokens within the prompts also translates to higher computational demands. Notably, numerous LLM web API services, including OpenAI APIs, charge based on the number of input tokens in the API calls. As such, we deem this metric valuable in assessing efficiency.
- *The average number of generated tokens outputted by LLMs per query.* Much like the assessment of average prompt tokens, this metric provides an evaluation of computational efficiency, but from a token generation perspective. Instead of focusing on the number of tokens in the prompt, it takes into account the number of tokens generated. This is particularly significant because transformer-based generative LLMs produce content token-by-token, with each subsequent token relying on the generation of preceding ones. Consequently, an increase in number of generated tokens leads to a corresponding increase in the computational cost, as each additional generated token implies another LLM forward inference. In fact, OpenAI applies a pricing structure wherein the cost for the number of generated tokens is twice that of the number of prompt tokens for their LLM APIs [1]. This underscores the substantial impact that generated tokens can have on computational expenses.
- *The average query latency.* We evaluate the run time efficiency of all the methods with average query latency. To conduct this assessment, a single GPU is employed, and queries are issued one at a time. The per-query latency is then averaged across all the queries in the dataset. It's important to highlight that for methods that support batching we always employ the maximum batch size to optimize GPU memory usage and parallel computation, thus maximizing efficiency for these particular methods. This approach ensures that the evaluation is conducted under conditions most favourable for efficiency gains. It is important to acknowledge that while other methods may not be able to use the batching strategy for individual queries, they do have the capability to utilize batching and parallel computing across various user queries in real-world scenarios. However, this lies more in engineering efforts and falls outside the scope of this paper: as such, we do not investigate this perspective.

## 4.2 Implementation details

To establish the initial BM25 first-stage ranking for all datasets, we employed the Pyserini Python library [13] with default settings. For LLM-based zero-shot re-rankers, we followed the prompts recommended in existing literature to guide Flan-t5 models of varying sizes (Flan-t5-large with 780M parameters, Flan-t5-xl with 3B parameters, and Flan-t5-xxl with 11B parameters) in executing the zero-shot ranking task.

Specifically, for the *pointwise.qlm* method, we adopted the prompt suggested by Sachan et al. [22]. For *pointwise.yes_no*, we use the prompt provided by Qin et al. [21]. For *listwise.generate*, we utilized the prompt designed by Sun et al. [23]. As for *pairwise.allpair*, *pairwise.heapsort*, and *pairwise.bubblesort*, we relied on the prompts from the original paper by Qin et al. [21]. For methods leveraging our *Setwise* prompting (i.e. *listwise.likelihood*, *setwise.heapsort*, and *setwise.bubblesort*), we employed the prompts detailed in Section 3.

In the case of *Listwise* approaches, we configure the window size ($w$) to contain 4 documents, each capped at a maximum of 100 tokens. The step size ($s$) is set to 2, and the number of repetitions ($r$) is set to 5. These settings take into account the token limitations imposed by Flan-t5 models, which have an input token cap of 512. A window size of 4 documents appears reasonable as it aligns well with the prompt capacity. Additionally, a step size of 2, combined with 5 repetitions, has theoretical guarantees of bringing the 10 most relevant documents to the top. For our *Setwise* approaches,

[1] https://openai.com/pricing, last visited 12 October 2023.

**Table 2: Results on TREC DL. All the methods re-rank BM25 top 100 documents. We present the ranking effectiveness in terms of NDCG@10, best values highlighted in boldface. Superscripts denote statistically significant improvements (paired Student's t-test with $p \leq 0.05$ with Bonferroni correction). #Inferences denotes the average number of LLM inferences per query. Pro. Tokens is the average number of tokens in the prompt for each query. Gen. tokens is the average number of generated tokens per query. Latency is the average query latency, in seconds.**

| | | | TREC DL 2019 | | | | | TREC DL 2020 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | Methods | NDCG@10 | #Inferences | Pro. tokens | Gen. tokens | Latency(s) | NDCG@10 | #Inferences | Pro. tokens | Gen. tokens | Latency(s) |
| | *a* | BM25 | .506 | - | - | - | - | .480 | - | - | - | - |
| Flan-t5-large | *b* | pointwise.qlm | .557 | 100 | 15211.6 | - | 0.6 | .567$^{a}$ | 100 | 15285.2 | - | 0.5 |
| | *c* | pointwise.yes_no | .654$^{abd}$ | 100 | 16111.6 | - | 0.6 | .615$^{ad}$ | 100 | 16185.2 | - | 0.6 |
| | *d* | listwise.generation | .561$^{a}$ | 245 | 119120.8 | 2581.35 | 54.2 | .547$^{a}$ | 245 | 119629.6 | 2460.1 | 52 |
| | *e* | listwise.likelihood | .669$^{abd}$ | 245 | 94200.7 | - | 10 | **.626**$^{abd}$ | 245 | 95208.3 | - | 10 |
| | *f* | pairwise.allpair | .666$^{abd}$ | 9900 | 3014383.1 | 49500 | 109.6 | .622$^{abd}$ | 9900 | 3014232.7 | 49500 | 108.9 |
| | *g* | pairwise.heapsort | .657$^{abd}$ | 230.3 | 104952.5 | 2303.3 | 16.1 | .619$^{abd}$ | 226.8 | 104242.1 | 2268.3 | 16.1 |
| | *h* | pairwise.bubblesort | .636$^{abd}$ | 844.2 | 381386.3 | 8441.6 | 58.3 | .589$^{ad}$ | 778.5 | 357358.5 | 7785.4 | 54.1 |
| | *i* | setwise.heapsort | .670$^{abd}$ | 125.4 | 40460.6 | 626.9 | 8.0 | .618$^{ad}$ | 124.2 | 40362.0 | 621.1 | 8.0 |
| | *j* | setwise.bubblesort | **.678**$^{abdh}$ | 460.5 | 147774.1 | 2302.3 | 29.1 | .624$^{abd}$ | 457.4 | 148947.3 | 2287.1 | 28.9 |
| Flan-t5-xl | *b* | pointwise.qlm | .542 | 100 | 15211.6 | - | 1.4 | .542$^{a}$ | 100 | 15285.2 | - | 1.4 |
| | *c* | pointwise.yes_no | .650$^{abd}$ | 100 | 16111.6 | - | 1.5 | .636$^{abd}$ | 100 | 16185.2 | - | 1.5 |
| | *d* | listwise.generation | .569$^{a}$ | 245 | 119163.0 | 2910 | 71.4 | .547$^{a}$ | 245 | 119814.3 | 2814.7 | 69 |
| | *e* | listwise.likelihood | .689$^{abd}$ | 245 | 94446.1 | - | 12.5 | .672$^{abd}$ | 245 | 95298.7 | - | 12.6 |
| | *f* | pairwise.allpair | **.713**$^{abcdehi}$ | 9900 | 2953436.2 | 49500 | 254.9 | .682$^{abcd}$ | 9900 | 2949457.6 | 49500 | 254.8 |
| | *g* | pairwise.heapsort | .705$^{abcd}$ | 241.9 | 110126.9 | 2418.6 | 20.5 | **.692**$^{abcdh}$ | 244.3 | 111341 | 2443.3 | 20.8 |
| | *h* | pairwise.bubblesort | .683$^{abd}$ | 886.9 | 400367.1 | 8869.1 | 75.1 | .662$^{abd}$ | 863.9 | 394954.2 | 8638.5 | 74.3 |
| | *i* | setwise.heapsort | .693$^{abcd}$ | 129.5 | 41665.7 | 647.4 | 9.6 | .678$^{abcd}$ | 127.8 | 41569.1 | 638.9 | 9.7 |
| | *j* | setwise.bubblesort | .705$^{abcd}$ | 466.9 | 149949.1 | 2334.5 | 35.2 | .676$^{abcd}$ | 463.5 | 151249.8 | 2317.6 | 35.3 |
| Flan-t5-xxl | *b* | pointwise.qlm | .506 | 100 | 15211.6 | - | 3.7 | .492 | 100 | 15285.2 | - | 3.7 |
| | *c* | pointwise.yes_no | .644$^{ab}$ | 100 | 16111.6 | - | 3.9 | .632$^{ab}$ | 100 | 16185.2 | - | 3.9 |
| | *d* | listwise.generation | .662$^{ab}$ | 245 | 119334.7 | 2824 | 100.1 | .637$^{ab}$ | 245 | 119951.6 | 2707.9 | 97.3 |
| | *e* | listwise.likelihood | .701$^{abcd}$ | 245 | 94537.5 | - | 36.6 | .690$^{abcd}$ | 245 | 95482.7 | - | 36.9 |
| | *f* | pairwise.allpair | .699$^{abcd}$ | 9900 | 2794942.6 | 49500 | 730.2 | .688$^{abcd}$ | 9900 | 2794928.4 | 49500 | 730.5 |
| | *g* | pairwise.heapsort | .708$^{abcdh}$ | 239.4 | 109402 | 2394 | 45 | **.699**$^{abcd}$ | 240.5 | 110211.8 | 2404.8 | 45.2 |
| | h | pairwise.bubblesort | .679$^{ab}$ | 870.5 | 394386 | 8705.3 | 162.5 | .681$^{abcd}$ | 842.9 | 387359.2 | 8428.5 | 158.8 |
| | i | setwise.heapsort | .706$^{abcd}$ | 130.1 | 42078.6 | 650.5 | 20.2 | .688$^{abcd}$ | 128.1 | 41633.7 | 640.6 | 20.0 |
| | *j* | setwise.bubblesort | **.711**$^{abcdh}$ | 468.3 | 150764.8 | 2341.6 | 72.6 | .686$^{abcd}$ | 467.9 | 152709.5 | 2339.6 | 73.2 |

we set the number of compared documents $c$ in each step to 3 for the main results. We further investigate the impact of $c$ in Section 5.3. For all other methods, we truncate the documents with the maximum number of tokens to 128.

We note that, among all the methods capable of utilizing both model output logits and generation outputs, we exclusively employ the latter. This choice is made in favor of a more general approach that allows for leveraging generation APIs across a wider range of closed-source LLMs. Nevertheless, we investigate the difference between using model output logits and generation outputs for our *Setwise* approaches in Section 5.1.

We carried out the efficiency evaluations on a local GPU workstation equipped with an AMD Ryzen Threadripper PRO 3955WX 16-Core CPU, a NVIDIA RTX A6000 GPU with 49GB of memory, and 128GB of DDR4 RAM.

# 5 RESULTS AND ANALYSIS

## 5.1 Effectiveness Results

Table 2 presents results for both ranking effectiveness and efficiency on TREC DL datasets.

In regards to ranking effectiveness, it is notable that all LLM-based zero-shot ranking approaches demonstrate a significant improvement over the initial BM25 ranking. The only exception to this trend is the *pointwise.qlm* approach on DL2019 across all models and DL2020 with the Flan-t5-xxl model. Interestingly, as the LLM size increases, the effectiveness of *pointwise.qlm* decreases. This finding is particularly unexpected, given the common assumption that larger LLMs tend to be more effective.

On the other hand, *pointwise.yes_no* method achieved a decent NDCG@10 score with Flan-t5-large when compared to other methods. However, effectiveness also did not increase as model size increased. These unexpected results for both *Pointwise* methods might be attributed to the requirement of a more refined model output calibration process, ensuring their suitability for comparison and sorting across different documents [21].

The *Listwise* approaches (*listwise.generation*) are far less effective when tested with Flan-t5-large and Flan-t5-xl. However, *listwise.generation* shows some improvement with Flan-t5-xxl. These results may be attributed to the fact that generating a ranking list requires fine-grained relevance preferences across multiple documents, a task that may exceed the capabilities of smaller models. In contrast, the *listwise.likelihood* approach, empowered by our *Setwise* prompt, markedly enhances the ranking effectiveness of the *Listwise* approach, even when utilizing smaller models. We acknowledge however that *listwise.likelihood* requires access to the model output logits, whereas *listwise.generation* does not. In the case of *Pairwise* and *Setwise* approaches, they consistently exhibit good ranking effectiveness across various model sizes and datasets.

**Table 3: Overall NDCG@10 obtained by methods on BEIR datasets. The best results are highlighted in boldface. Superscripts denote statistically significant improvements (paired Student's t-test with $p \leq 0.05$ with Bonferroni correction).**

| | # | Methods | Covid | NFCorpus | Touche | DBPedia | SciFact | Signal | News | Robust04 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *a* | BM25 | .595 | .322 | .442 | .318 | .679 | .331 | .395 | .407 | .436 |
| Flan-t5-large | *b* | pointwise.qlm | .664$^{a}$ | .322 | .260 | .305 | .644$^{c}$ | .314$^{c}$ | .413$^{c}$ | .439$^{af}$ | .420 |
| | *c* | pointwise.yes_no | .664$^{a}$ | .308 | .238 | .296 | .504 | .275 | .346 | .456$^{af}$ | .386 |
| | *d* | listwise.generation | .692$^{a}$ | .333$^{ac}$ | .441$^{bcefhi}$ | .391$^{abc}$ | .650$^{c}$ | .343$^{ace}$ | .428$^{ac}$ | .441$^{af}$ | .465 |
| | *e* | listwise.likelihood | .756$^{abcd}$ | .334$^{c}$ | .327$^{bc}$ | **.444**$^{abcdfgh}$ | .639$^{c}$ | .308$^{c}$ | **.453**$^{ac}$ | .475$^{abdfg}$ | .467 |
| | *f* | pairwise.heapsort | .761$^{abcdg}$ | .336$^{c}$ | .318$^{bc}$ | .414$^{abcd}$ | .671$^{chi}$ | .325$^{c}$ | .440$^{ac}$ | .402 | .458 |
| | *g* | pairwise.bubblesort | .714$^{a}$ | **.341**$^{abcdh}$ | **.447**$^{bcefhi}$ | .416$^{abcd}$ | **.700**$^{bcdefhi}$ | **.361**$^{abcdefh}$ | .440$^{ac}$ | .439$^{af}$ | .482 |
| | *h* | setwise.heapsort | **.768**$^{abcdg}$ | .325$^{c}$ | .303$^{c}$ | .413$^{abcd}$ | .620$^{c}$ | .319$^{c}$ | .439$^{c}$ | .462$^{abf}$ | .456 |
| | *i* | setwise.bubblesort | .761$^{abcdg}$ | .338$^{abch}$ | .394$^{bcefh}$ | .441$^{abcdfgh}$ | .636$^{c}$ | .351$^{bcefh}$ | .447$^{ac}$ | **.497**$^{abcdefgh}$ | **.483** |
| Flan-t5-xl | *b* | pointwise.qlm | .679$^{a}$ | .330 | .216 | .310$^{c}$ | .696$^{c}$ | .299 | .422 | .427 | .422 |
| | *c* | pointwise.yes_no | .698$^{a}$ | .331 | .269 | .273 | .553 | .297 | .413 | .479$^{ab}$ | .414 |
| | *d* | listwise.generation | .650$^{a}$ | .334$^{a}$ | **.451**$^{bcefghi}$ | .366$^{abc}$ | .694$^{c}$ | .349$^{abcefh}$ | .437$^{a}$ | .475$^{ab}$ | .470 |
| | *e* | listwise.likelihood | .736$^{abd}$ | **.360**$^{abcd}$ | .310$^{b}$ | **.449**$^{abcdfghi}$ | .686$^{c}$ | .320 | .472$^{abc}$ | .526$^{abcd}$ | .482 |
| | *f* | pairwise.heapsort | **.778**$^{abcde}$ | .355$^{abcd}$ | .303$^{b}$ | .417$^{abcd}$ | .711$^{ch}$ | .317 | .471$^{abc}$ | .550$^{abcdehi}$ | .488 |
| | *g* | pairwise.bubblesort | .763$^{abcd}$ | .359$^{abcd}$ | .400$^{bcefhi}$ | .432$^{abcdf}$ | **.734**$^{abcdefhi}$ | .353$^{bcefh}$ | .485$^{abcd}$ | **.553**$^{abcdehi}$ | **.510** |
| | *h* | setwise.heapsort | .757$^{abcd}$ | .352$^{abcd}$ | .283$^{b}$ | .428$^{abcdf}$ | .677$^{c}$ | .314 | .465$^{ac}$ | .520$^{abcd}$ | .475 |
| | *i* | setwise.bubblesort | .756$^{abcd}$ | .353$^{abcd}$ | .330$^{bch}$ | .438$^{abcdfh}$ | .691$^{c}$ | **.362**$^{abcefh}$ | **.497**$^{abcd}$ | .537$^{abcdh}$ | .496 |
| Flan-t5-xxl | *b* | pointwise.qlm | .707$^{a}$ | .342$^{ac}$ | .188 | .324 | .712$^{c}$ | .307$^{c}$ | .431 | .440$^{a}$ | .431 |
| | *c* | pointwise.yes_no | .691$^{a}$ | .322 | .240$^{b}$ | .305 | .623 | .274 | .392 | .515$^{ab}$ | .420 |
| | *d* | listwise.generation | .664$^{a}$ | .344$^{ac}$ | **.453**$^{bcefhi}$ | **.441**$^{abcefghi}$ | .736$^{ac}$ | .353$^{bcefh}$ | .458$^{ac}$ | .495$^{ab}$ | .493 |
| | *e* | listwise.likelihood | .749$^{acd}$ | .352$^{ac}$ | .307$^{bc}$ | .416$^{abch}$ | .725$^{ac}$ | .316$^{c}$ | .479$^{abc}$ | .518$^{abd}$ | .483 |
| | *f* | pairwise.heapsort | .738$^{acd}$ | .359$^{abcdhi}$ | .324$^{bc}$ | .407$^{abc}$ | .744$^{abc}$ | .328$^{c}$ | .487$^{abc}$ | .543$^{abcdeh}$ | .491 |
| | *g* | pairwise.bubblesort | .733$^{ad}$ | **.363**$^{abcdehi}$ | .423$^{bcefh}$ | .421$^{abcfh}$ | **.756**$^{abcdeh}$ | **.355**$^{bcefh}$ | **.490**$^{abcd}$ | **.550**$^{abcdehi}$ | **.511** |
| | *h* | setwise.heapsort | .752$^{abcd}$ | .346$^{ac}$ | .297$^{bc}$ | .402$^{abc}$ | .726$^{ac}$ | .321$^{c}$ | .473$^{abc}$ | .513$^{ab}$ | .479 |
| | *i* | setwise.bubblesort | **.768**$^{abcdfg}$ | .346$^{ac}$ | .388$^{bcefh}$ | .424$^{abcfh}$ | .754$^{abceh}$ | .343$^{bceh}$ | .479$^{abc}$ | .534$^{abdeh}$ | .505 |

In Table 3, we present the zero-shot ranking effectiveness of all methods (with the exception of *pairwise.allpair* due to its computationally intensive nature) across 8 widely-used BEIR datasets. Notably, we identify several different trends that deviate from observations made on the TREC DL datasets.

Firstly, *pointwise.qlm* exhibits a slightly higher average NDCG@10 score compared to *pointwise.yes_no*. Moreover, the effectiveness of *pointwise.qlm* remains stable even as the model size increases. Secondly, *listwise.generation* demonstrates comparable effectiveness to *listwise.likelihood*, with the majority of gains obtained in the Touche dataset, where other methods perform worse. Lastly, both *Pairwise* and *Setwise* methods that leverage the bubble sort algorithm consistently demonstrate higher average NDCG@10 compared to when they utilize the heap sort algorithm, regardless of the model size. Overall, the variety of results we observe across different experimental settings shows the importance of not drawing conclusions about effectiveness from single datasets or model sizes.

We note that if the LLM output logits are accessible, our *Setwise* approaches can also utilize these logits to estimate the likelihood of the most relevant document label. This approach eliminates the need for token generation, requiring only a single LLM forward inference to yield the output results, thus avoiding the generation of unexpected tokens. Surprisingly, in our experiments we find that using model logits for our *Setwise* approaches resulted in no change in ranking effectiveness when compare to generation, suggesting that the inference of our *Setwise* approaches that fully relies on token generation is very robust.

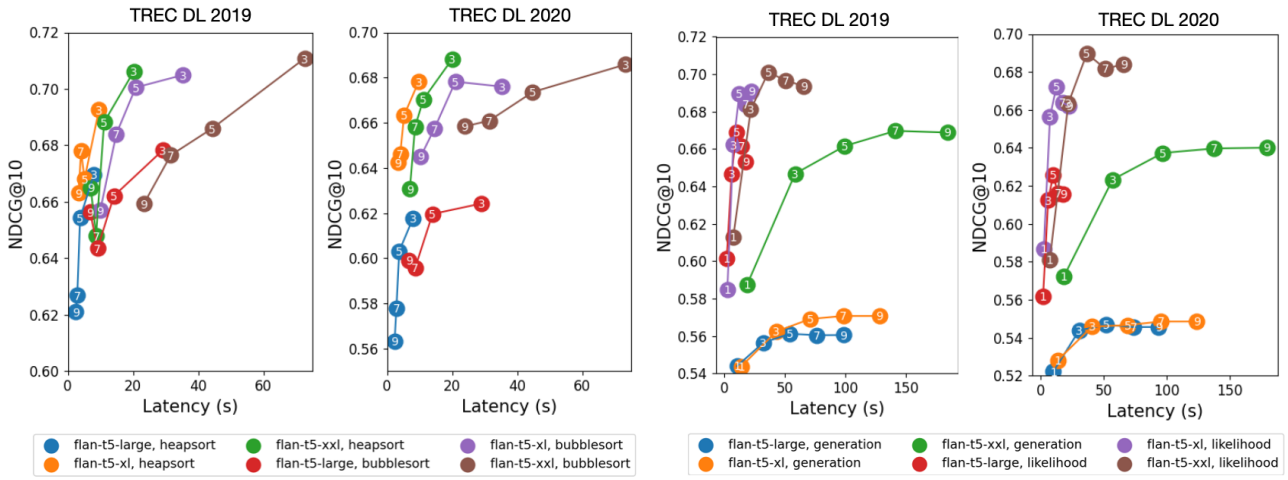
## 5.2 Efficiency Results

Regarding computational and runtime efficiency, the results presented in Table 2 indicate that both *Pointwise* methods exhibit fewest inference, prompt tokens, and no generated tokens. Furthermore, their computational efficiency and query latency are optimized due to efficient GPU-based batched inference. It is worth noting, however, that these methods do come with certain limitations. Specifically, they require access to the model output logits (thus currently limiting their use to just open source LLMs) and are less effective when used with larger models. In contrast, *pairwise.allpair* appears to be the most expensive method that consumes the most number of prompt tokens and generated tokens due to the large number of document pair preferences needed to be inferred. Hence, even with GPU batching, *pairwise.allpair* still has the worst query latency. In contrast, approaches utilizing our *Setwise* prompting—namely, *listwise.likelihood*, *setwise.heapsort*, and *setwise.bubblesort*, are far more efficient than their counterparts, *listwise.generate*, *pairwise.heapsort*, and *pairwise.bubblesort* respectively. Notably, these improvements are achieved without compromising effectiveness. Section 5.3 will discuss further approaches on improving efficiency.

The *setwise.bottlesort* and *pairwise.heapsort* methods show comparable NDCG@10, but *pairwise.heapsort* is cheaper. On the other hand, our *setwise.heapsort* provides a reduction of $\approx 62\%$ in cost by only marginally reducing NDCG@10 (a 0.8% loss).

## 5.3 Effectiveness and Efficiency Trade-offs

Our *Setwise* prompting is characterized by a hyperparameter $c$ controlling the number of compared documents within the prompt




**(a) Setwise** **(b) Listwise**

**Figure 3: Effectiveness and efficiency trade-offs offered by different approaches. (a – Setwise): The numbers in the scatter plots represent the number of compared documents $c$ at each step of the sorting algorithm. (b – Listwise) The numbers in the scatter plots represent the number of sliding windows repetitions $r$.**

for each step in the sorting algorithms. In the previous experiments, we always set $c = 3$. Adjusting this hyperparameter allows one to further enhance efficiency by incorporating more compared documents into the prompt, thereby reducing the number of LLM inference calls. However, we acknowledge that there is an input length limitation to LLMs (in our experiments this is 512 prompt tokens) and setting $c$ to a large value may require more aggressive document truncation, likely impacting effectiveness.

To investigate the trade-off between effectiveness and efficiency inherent in our *Setwise* approach, we set $c = 3, 5, 7, 9$ while truncating the documents in the prompt to 128, 85, 60, 45 tokens, respectively. This reduction in document length is necessary to ensure prompt size is not exceeded. The NDCG@10, along with query latency for all models while varying $c$, is visualized in Figure 3a for the TREC DL datasets. As expected, larger $c$ reduces query latency but often degrades effectiveness. Notably, the heap sort algorithm consistently proves more efficient than bubble sort. For instance, with Flan-t5-xl and $c = 9$, heap sort achieves strong NDCG@10 with a query latency of ≈3 seconds. When compared to the other methods outlined in Table 2, this represents the lowest query latency, except for the *Pointwise* approaches with Flan-t5-large, albeit with superior ranking effectiveness. It's worth noting that the ranking effectiveness decline with larger $c$ values could also be attributed to the increased truncation of passages. LLMs with extended input length capacity might potentially yield improved ranking effectiveness for larger $c$. This area warrants further exploration in future studies.

Similarly, the *Listwise* balance effectiveness and efficiency through the adjustment of the repetition count $r$ for the sliding window. In our prior experiments, we consistently set $r = 5$ to ensure that at least 10 of the most relevant documents can be brought to the top. In Figure 3b, we investigate the influence of varying $r$ on *Listwise* approaches. Latency exhibits a linear relationship with $r$, which aligns with expectations. A larger value of $r$ can enhance the effectiveness of *listwise.generate*, and beyond $r > 5$, the improvement levels off. Conversely, the *listwise.likelihood* approach, which leverages our *Setwise* prompting, showcases notably higher effectiveness and efficiency. Even with a small value of $r$ the performance of *listwise.likelihood* exceeds that of *listwise.generate*, with the highest performance achieved around $r = 5$.

## 5.4 Sensitivity to the Initial Ranking

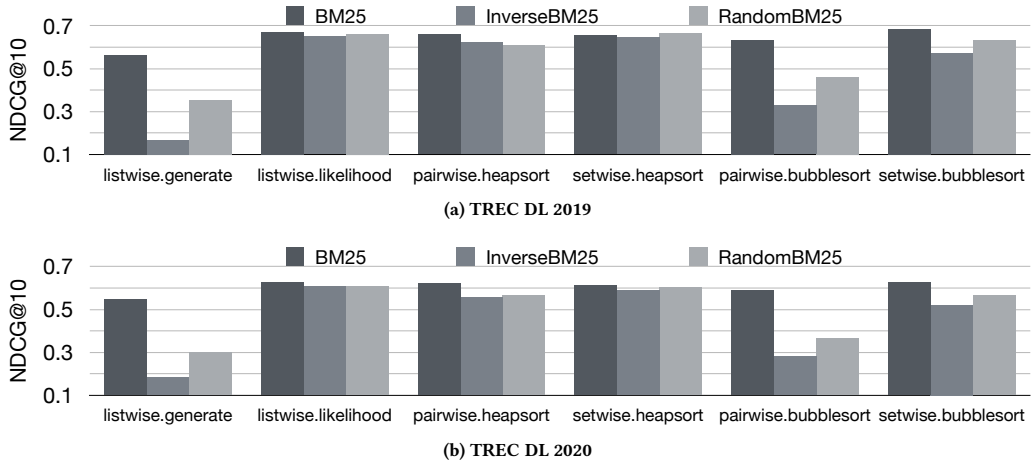
The ranking effectiveness of the original *Listwise* and *Pairwise* methods is influenced by the initial ranking order [21, 23]. To investigate this aspect in relation to our approach, we consider different orderings of the initial BM25 list; specifically, 1) initial BM25 ranking; 2) inverted BM25 ranking; and 3) random shuffled BM25 ranking. Each of these initial rankings was used to test different reranking methods using Flan-t5-large. The results are presented in Figure 4. Different initial ranking orders negatively impact *listwise.generate*, *pairwise.heapsort* and *pairwise.bubblesort*; *pairwise.heapsort* is the most robust method. These findings align with the literature [21, 23].

In contrast, *Setwise* prompting is far more robust to variations in the initial ranking order. Both *listwise.likelihood* and *setwise.bubblesort* exhibit large improvements over *listwise.generate* and *pairwise.bubblesort*, in the case of the inverted BM25 ranking and randomly shuffled BM25 ranking. Moreover, they demonstrate a similar level of robustness to *pairwise.heapsort*. This leads us to the conclusion that our *Setwise* prompting approach substantially enhances the zero-shot re-ranking with LLMs in relation to the initial ranking.

## 5.5 Effectiveness and Costs of other LLMs

In the previous sections, we only used Flan-T5 as the backbone LLM. Flan-T5 is a transformer encoder-decoder model. In this section, to better understand the impact of different models, we investigate popular transformer decoder-only LLMs (open-sourced: llama2-chat-7b[2] [28], vicuna-13b-v1.5[3] [35]; closed-source: OpenAI gpt-3.5-turbo-1106[4]) on DL2019 and DL2020. For open-source models

[2]https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[3]https://huggingface.co/lmsys/vicuna-13b-v1.5
[4]https://platform.openai.com/docs/models/gpt-3-5




**(a) TREC DL 2019**

**(b) TREC DL 2020**

**Figure 4: Sensitivity to the initial ranking. We use Flan-t5-large and $c = 4$ for the Setwise approach.**

**Table 4: Results obtained with other LLMs on TREC DL datasets: We report the cost in terms of query latency in seconds (s) if the LLM's weights are publicly available, or the API call costs in US dollars ($) if the LLM is only accessible via API calls.**

| | | | TREC DL 2019 | | TREC DL 2020 | |
|---|---|---|---|---|---|---|
| | # | Methods | NDCG@10 | Cost(s or $) | NDCG@10 | Cost(s or $ ) |
| llama2-chat-7b | *a* | listwise.generation | .508 | 144.9 s | .475$^{c}$ | 143.9 s |
| | *b* | pairwise.bubblesort | .538$^{ac}$ | 64.3 s | .503$^{ac}$ | 61.5 s |
| | *c* | pairwise.heapsort | .465 | 18.3 s | .416 | 17.9 s |
| | *d* | setwise.bubblesort | **.578**$^{abc}$ | 38.2 s | **.530**$^{ac}$ | 38.5 s |
| | *e* | setwise.heapsort | .568$^{c}$ | **10.9** s | .514$^{c}$ | **10.9** s |
| vicuna-13b | *a* | listwise.generation | .639 | 225.9 s | **.608** | 226.3 s |
| | *b* | pairwise.bubblesort | .612 | 166.6 s | .592 | 169.0 s |
| | *c* | pairwise.heapsort | .617 | 46.0 s | .582 | 45.6 s |
| | *d* | setwise.bubblesort | .622 | 57.6 s | .602 | 58.8 s |
| | *e* | setwise.heapsort | **.659**$^{bcd}$ | **16.7** s | .583 | **16.4** s |
| gpt-3.5 | *a* | listwise.generation | **.712** | 0.045 $ | **67.2** | 0.046 $ |
| | *b* | pairwise.heapsort | .694 | 0.171 $ | 65.1 | 0.169 $ |
| | *c* | setwise.bubblesort | .699 | 0.084 $ | 65.9 | 0.088 $ |
| | *d* | setwise.heapsort | .693 | **0.029** $ | 65.6 | **0.028** $ |

we measure cost in terms of query latency in seconds (s); for closed models we measure the API call costs in US dollars ($).[5] Results are presented in the Table 4.

Flan-T5 in our previous results is a better backbone than Llama2 and Vicuna, regardless of ranking method. Notably, *Setwise* exhibits the best overall performance when considering these models, showcasing its robustness. For gpt-3.5-turbo (closed model), we compared *Setwise* and *Listwise* using 10 documents at the time ($c = 10$), as this LLM has a longer input context limit; for *Listwise*, we set window size 10, step size of 5 and repeat sorting twice for fair comparison with *Setwise*. *Listwise* achieves the highest effectiveness; however, *Setwise* achieves similar effectiveness (no significant differences) but at only half the cost.

We note that there could be potential data contamination with instruction-tuned LLMs: during the instruction fine-tuning tasks, these models could have been fine-tuned on the MS MARCO or BEIR datasets. Although the document ranking task and the ranking prompts used in this paper are unlikely to be part of the instruction fine-tuning dataset used for these LLMs, we acknowledge that data contamination could still artificially impact the effectiveness of the considered LLMs in ranking tasks. However, since we conducted experiments with different methods based on the same instruction-tuned model, we believe our empirical comparison still offers insights for the practical use of LLM-based re-rankers.

[5]We exclude *Pointwise* approaches as they are less effective, and *pairwise.bubblesort* for gpt-3.5-turbo as it is too costly.

## 6 CONCLUSION

We undertook a comprehensive study of existing LLM-based zero-shot document ranking methods, employing strict and consistent experimental conditions. Our primary emphasis was on evaluating both their ranking effectiveness and their efficiency in terms of computational efficiency and runtime latency — factors that are often disregarded in previous studies. Our findings unveil some unforeseen insights, and effectiveness-efficiency trade-offs between different methods. This information equips practitioners with valuable guidance when selecting the most appropriate method for their specific applications.

To further boost efficiency of LLM-based zero-shot document ranking, we introduced an innovative *Setwise* prompting strategy. *Setwise* has the potential to enhance both effectiveness and efficiency for *Listwise* approaches provided the model logits are accessible. *Setwise* also notably enhances the efficiency of sorting-based *Pairwise* approaches. Furthermore, *Setwise* prompting offers a straightforward way to balance effectiveness and efficiency by incorporating more documents for comparison in the prompt. Additionally, approaches equipped with *Setwise* prompting demonstrated strong robustness to variation in the initial retrieval set used for reranking.

Future work should focus on evaluating the *Setwise* prompting approach on a wider array of LLMs, including LLaMA models [26, 27] as well as the OpenAI LLM APIs. Additionally, recent advanced self-supervised prompt learning techniques [7, 34] could be used to refine the *Setwise* approach.

# REFERENCES


[1] Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. 2022. Large language models are zero-shot clinical information extractors. *arXiv preprint arXiv:2205.12689* (2022).

[2] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders. arXiv:2404.05961 [cs.CL]

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[5] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. *arXiv preprint arXiv:2102.07662* (2021).

[6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the TREC 2019 deep learning track. *arXiv preprint arXiv:2003.07820* (2020).

[7] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution. *arXiv preprint arXiv:2309.16797* (2023).

[8] Lukas Gienapp, Maik Fröbe, Matthias Hagen, and Martin Potthast. 2022. Sparse Pairwise Re-Ranking with Pre-Trained Transformers. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval* (Madrid, Spain) *(ICTIR '22)*. ACM, New York, NY, USA, 72–80. https://doi.org/10.1145/3539813.3545140

[9] Donald Ervin Knuth. 1997. *The art of computer programming*. Vol. 3. Pearson Education.

[10] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[11] Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Prateek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftekhar Naim. 2024. Gecko: Versatile Text Embeddings Distilled from Large Language Models. arXiv:2403.20327 [cs.CL]

[12] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110* (2022).

[13] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. ACM, New York, NY, USA, 2356–2362. https://doi.org/10.1145/3404835.3463238

[14] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. Pretrained Transformers for Text Ranking: BERT and Beyond. arXiv:2010.06467 [cs.IR]

[15] 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).

[16] Aliaksei Mikhailiuk, Clifford Wilmot, Maria Perez-Ortiz, Dingcheng Yue, and Rafal Mantiuk. 2021. Active Sampling for Pairwise Comparisons via Approximate Message Passing and Information Gain Maximization. In *2020 IEEE International Conference on Pattern Recognition (ICPR)*.

[17] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 708–718.

[18] Jay M Ponte and W Bruce Croft. 2017. A language modeling approach to information retrieval. In *ACM SIGIR Forum*, Vol. 51. ACM New York, NY, USA, 202–208.

[19] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *arXiv preprint arXiv:2101.05667* (2021).

[20] 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).

[21] 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).

[22] 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 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 3781–3797. https://doi.org/10.18653/v1/2022.emnlp-main.249

[23] 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. *arXiv preprint arXiv:2304.09542* (2023).

[24] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[25] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. 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)*.

[26] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[28] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).

[29] 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 *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[30] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2024. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv:2212.03533 [cs.CL]

[31] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving Text Embeddings with Large Language Models. arXiv:2401.00368 [cs.CL]

[32] Shuai Wang, Harrisen Scells, Bevan Koopman, and Guido Zuccon. 2023. Can ChatGPT Write a Good Boolean Query for Systematic Review Literature Search?. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. ACM, New York, NY, USA, 1426–1436. https://doi.org/10.1145/3539618.3591703

[33] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*.

[34] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409* (2023).

[35] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL]

[36] Shengyao Zhuang, Hang Li, and Guido Zuccon. 2021. Deep query likelihood model for information retrieval. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28–April 1, 2021, Proceedings, Part II 43*. Springer, 463–470.

[37] 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. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 8807–8817. https://doi.org/10.18653/v1/2023.findings-emnlp.590

[38] Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. 2024. PromptReps: Prompting Large Language Models to Generate Dense and Sparse Representations for Zero-Shot Document Retrieval. arXiv:2404.18424 [cs.IR]

[39] Shengyao Zhuang and Guido Zuccon. 2021. TILDE: Term independent likelihood moDEl for passage re-ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1483–1492.