LLM Rankers

Ashwin Mathur* Varun Mathur*
{ashwinxmathur, varunm500}@gmail.com

February 8, 2025

Abstract

Large Language Models (LLMs) can be efficiently used for document ranking in information retrieval tasks. Compared to traditional methods, LLMs provide more accurate and contextually relevant rankings. The zero-shot capability of LLMs allows them to be used for ranking without extensive training on specific datasets. We release implementations for the three types of LLM based ranking techniques: Pairwise (Qin et al. 2023), Setwise (S. Zhuang et al. 2023), and Listwise (Ma, X. Zhang, et al. 2023a). We evaluated the rankers using pipelines built with the Haystack framework. The evaluation was conducted on the *FIQA*, *SciFact*, *NFCorpus*, *TREC-19* and *TREC-20* datasets. The *Mistral*, *Phi-3*, and *Llama-3* models were paired with Pairwise and Setwise rankers. The *RankLlama* and *RankZephyr* models were paired with the Listwise Ranker. All rankers scored very closely across all datasets. The *RankLlama* and *RankZephyr* (used with the Listwise ranker) consistently achieved slightly better results than the other rankers. Among the models not explicitly trained for reranking, the Llama-3 model with the Setwise and Pairwise ranker performed the best.

1 Introduction

Large Language Models (LLMs) have emerged as powerful tools for document ranking tasks, showcasing their remarkable ability to understand and rank documents without any task-specific training. Our work focuses on implementing the Pairwise, Setwise, and Listwise rankers as well as evaluating these rankers on diverse tasks.

2 Ranking Techniques

Ranking strategies for utilizing LLMs in ranking tasks can be categorized into four main approaches: Pointwise, Pairwise, Listwise, and Setwise Ranking.

Given a query q and a set of candidate items $D = d_1, d_2, \ldots, d_n$, the objective is to determine the ranking of these candidates, represented as $R = r_1, r_2, \ldots, r_n$. Here, $r_i \in 1, 2, \ldots, n$ denotes the rank of the candidate d_i. For example, $r_i = 3$, means that the document d_i is ranked third among the n candidates. A ranking model $f(.)$ assigns scores to the candidates based on their relevance to the query: $s_i = f(q, d_i)$, and the candidates are then ranked according to these relevance scores: $r_i = argsort_i(s_1, s_2, \ldots, s_n)$.

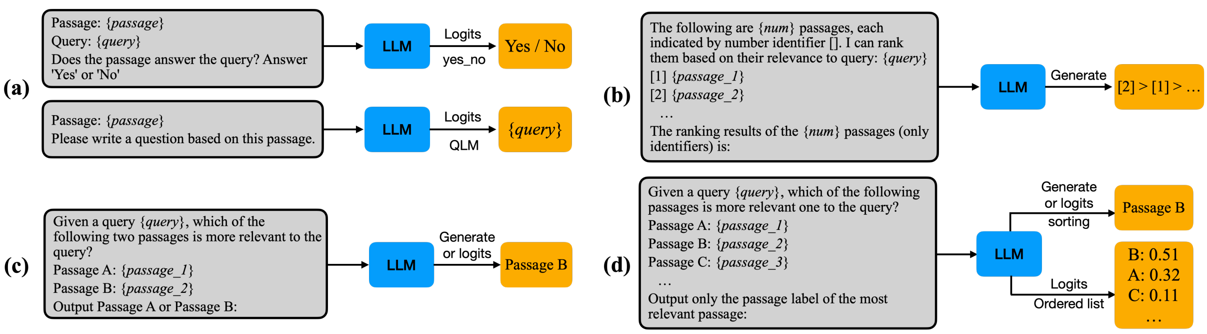

Figure 1: Different ranking strategies, (a) Pointwise, (b) Listwise, (c) Pairwise and (d) Setwise, figure taken from (S. Zhuang et al. 2023)

.

*Equal Contribution

2.1 Log Probabilities

Large language models (LLMs) are trained to predict the likelihood of sequences of tokens (e.g., words, characters) based on the given context. Log probabilities (OpenAI 2023) are a way to express the likelihood of certain tokens or sequences of tokens occurring in the output of these models. Log probabilities of output tokens indicate the likelihood of each token occurring in the sequence given the context.

To compute the log probability of a sequence, we follow these steps:

1. Retrieve the logits for each token in the sequence.
2. Apply the softmax function to convert logits into probabilities.
3. Calculate the log probabilities of each token.
4. Sum the log probabilities to get the log probability of the entire sequence.

Let's denote the tokens in the sequence as t_1, t_2, t_3, t_4. If the probabilities of these tokens given the previous context are $p(t_1), p(t_2|t_1), p(t_3|t_1, t_2)$, and $p(t_4|t_1, t_2, t_3)$ respectively, then the log probability of the sequence is:

$$\log(p(t_1, t_2, t_3, t_4)) = \log(p(t_1)) + \log(p(t_2|t_1)) + \log(p(t_3|t_1, t_2)) + \log(p(t_4|t_1, t_2, t_3))$$

2.1.1 Difference between Logits and Log Probabilities

- **Logits** are the raw, unnormalized scores output by a model before they are converted into probabilities. These scores are transformed using the Softmax function to produce a probability distribution over the possible next tokens.
- The **log probability** of a token is obtained by taking the natural logarithm of its probability. If p_i is the probability of the i-th token, then its log probability is $\log(p_i)$. Since probabilities range from 0 to 1, log probabilities range from $-\infty$ to 0.

2.1.2 Advantages of using Log Probabilities

- **Numerical Stability**: Working with log probabilities enhances numerical stability, especially when dealing with very small probabilities. This is crucial for avoiding underflow issues in computations.
- **Additivity:** One of the key advantages of using log probabilities is that they are additive. The log probability of a sequence of tokens is simply the sum of the log probabilities of the individual tokens:

  $$\log(p(\text{sequence})) = \sum_{i=1}^{n} \log(p_i)$$

  This property simplifies the computation of the joint probability of a sequence.
- **Comparative Ranking:** Log probabilities allow for easy comparison between different sequences. Higher log probabilities (closer to 0) indicate higher likelihoods. This enables the ranking of sequences based on their likelihood.

Log probabilities play a crucial role in various natural language processing (NLP) tasks involving Large Language Models (LLMs). Here are some key use cases of log probabilities:

- **Scoring Outputs:** To rank different outputs generated by an LLM, we calculate the log probability of each output sequence. The sequence with the highest log probability is considered the most likely and thus ranked highest.

  Given a sequence of tokens $t_1, t_2, \ldots, t_n$, the log probability of the sequence is:

  $$\log(p(\text{sequence})) = \sum_{i=1}^{n} \log(p(t_i \mid t_1, t_2, \ldots, t_{i-1}))$$

  This summation leverages the chain rule of probability, where each token's probability is conditioned on all previous tokens.

- **Average Per-Token Log Probability:** To normalize the log probability by the length of the sequence, we calculate the average log probability per token. This is particularly useful for comparing sequences of different lengths.

$$\text{Average log probability} = \frac{1}{n}\sum_{i=1}^{n}\log(p(t_i \mid t_1, t_2, \ldots, t_{i-1}))$$

  This metric ensures that longer sequences are not unfairly penalized and provides a normalized measure of likelihood.

- **Confidence Measurement:** Log probabilities offer a quantitative measure of the model's confidence in its predictions. By examining the log probabilities of the top candidate tokens at each position, we can gauge the model's certainty about its choices.

  For a given token position j, the model generates a set of candidate tokens $\{t_{j1}, t_{j2}, \ldots, t_{jk}\}$ with corresponding log probabilities $\{\log(p(t_{j1})), \log(p(t_{j2})), \ldots, \log(p(t_{jk}))\}$. The difference in log probabilities among these candidates indicates the model's confidence.

- **Classification Tasks**: In classification tasks, log probabilities help set confidence thresholds. By associating log probabilities with each class prediction, users can determine the required confidence level for making a classification decision.

  Given a set of classes $C = \{c_1, c_2, \ldots, c_m\}$ with associated log probabilities $\{\log(p(c_1)), \log(p(c_2)), \ldots, \log(p(c_m))\}$, the class with the highest log probability is chosen:

$$\hat{c} = \arg\max_{c_i \in C} \log(p(c_i))$$

  A threshold τ can be applied to $\log(p(\hat{c}))$ to decide if the confidence in $\hat{c}$ is sufficient for classification.

- **Retrieval and Q&A Evaluation:** In retrieval-based applications, log probabilities assist in self-evaluation. For instance, in a question-answering system, the model can use log probabilities to assess if the retrieved content sufficiently answers the query.

  If the model retrieves a set of passages $\{P_1, P_2, \ldots, P_k\}$ and generates answers $\{A_1, A_2, \ldots, A_k\}$ with corresponding log probabilities, the evaluation metric can be:

$$\log(p(\text{answer is correct})) = \log(p(A_j \mid P_j))$$

- **Autocomplete and Token Highlighting:** In autocomplete applications, log probabilities inform the likelihood of each possible next token, aiding in suggestion generation. Token highlighting leverages log probabilities to emphasize tokens with higher likelihoods.

  For a partially typed sequence $t_1, t_2, \ldots, t_i$, the model predicts the next token t_{i+1} with log probabilities for top candidates $\{t_{i+1,1}, t_{i+1,2}, \ldots, t_{i+1,k}\}$:

$$\{\log(p(t_{i+1,1} \mid t_1, t_2, \ldots, t_i)), \log(p(t_{i+1,2} \mid t_1, t_2, \ldots, t_i)), \ldots\}$$

  Highlighting can be based on these log probabilities to visually represent the model's predictions.

- **Calculating Perplexity**: Perplexity is a measure of how well a probability distribution or model predicts a sample. It is commonly used to evaluate language models. The perplexity PP of a sequence $t_1, t_2, \ldots, t_n$ is given by:

$$PP = \exp\left(-\frac{1}{n}\sum_{i=1}^{n}\log(p(t_i \mid t_1, t_2, \ldots, t_{i-1}))\right)$$

Log probabilities are a versatile tool in NLP, enabling precise probability calculations, enhancing numerical stability, and facilitating various applications, from output ranking to confidence measurement and classification. By understanding and leveraging log probabilities, we can significantly improve the performance and reliability of LLM-based applications.

2.2 Pointwise Ranking

In the Pointwise ranking method, the reranker takes both the query and a candidate document to directly generate a relevance score. These independent scores assigned to each document d_i are then used to reorder the candidate set D.

This method can be further classified into two popular approaches based on how the relevance score is calculated. It is calculated based on how likely the document is relevant to the query or how likely the query can be generated from the document.

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.

2.2.1 Instructional Relevance Generation

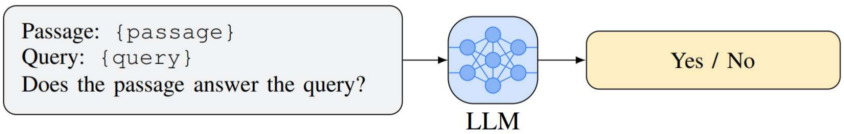

Figure 2: The Pointwise relevance generation approach, figure taken from (Qin et al. 2023).

In Instructional Relevance Generation (Liang et al. 2022), the LLMs are prompted to output either "Yes" or "No" to determine the relevance of the candidates to a given query.

The generation probability is then converted to the relevance score:

$$s_i = \begin{cases} 1 + f(\text{Yes}|I_{\text{RG}}(q, d_i)), & \text{if output Yes} \\ 1 - f(\text{No}|I_{\text{RG}}(q, d_i)), & \text{if output No} \end{cases}$$

Here $f(.)$ represents the large language model, and I_{RG} denotes the relevance generation instruction that converts the input q and d_i into the text-based prompt.

2.2.2 Instructional Query Generation

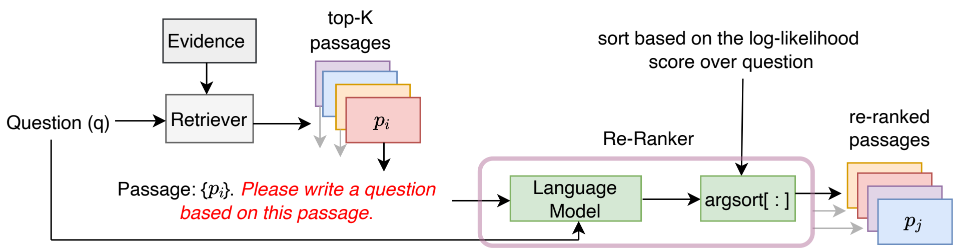

Figure 3: The LLM scores are used for passage reordering in the Instructional Query Generation approach, figure taken from (Sachan, Lewis, Joshi, et al. 2022).

Instructional Query Generation (Sachan, Lewis, Joshi, et al. 2022) uses LLMs to generate a query based on the document and to measure the probability of generating the actual query. This is done by employing a zero-shot ranking approach using a LLM. A natural language instruction "Please write a question based on this passage" is appended to the document.

The likelihood of tokens generated to be the query is calculated given the document:

$$\log p(q|d_i) = \frac{1}{|q|} \sum_t \log p(d_t|q_{<t}, d_i; \Theta)$$

where Θ denotes the LLM parameters, and $|q|$ denotes the number of query tokens. The candidate set of documents is then sorted based on $\log p(q|z)$.

2.3 Pairwise Ranking

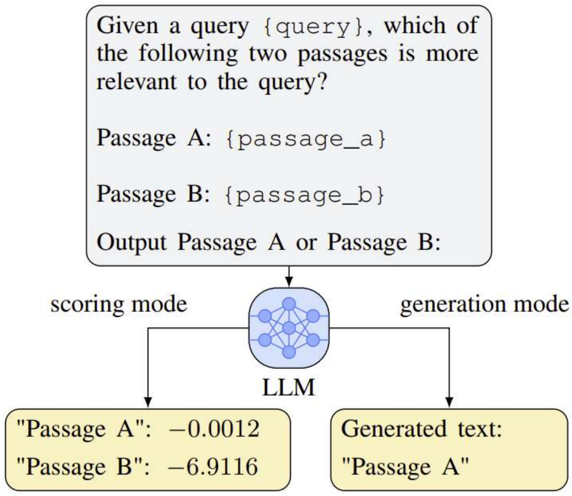

Figure 4: The Pairwise Ranking Prompting approach, figure taken from (Qin et al. 2023).

In Pairwise Ranking Prompting (PRP) (Qin et al. 2023), a pair of candidate items (d_i, d_j) along with the user query (q) serve as prompts to guide the LLMs to determine which document is the most relevant to the given query.

$$c_{i,j} = \begin{cases} 1, & \text{if } f(I_{\text{PRP}}(q, d_i, d_j)) = i \\ 0, & \text{if } f(I_{\text{PRP}}(q, d_i, d_j)) = j \\ 0.5, & \text{else} \end{cases}$$

Here, $c_{i,j}$ denotes the choice of LLM $f(.)$, and I_{PRP} is a specific pairwise comparison instruction employed to instruct the LLM. This approach usually consults the LLM twice (with $I_{PRP}(q, d_i, d_j)$ and $I_{PRP}(q, d_j, d_i)$) for every pair d_i and d_j because LLMs exhibit sensitivity to the order of the text in the prompt.

Subsequently, to compute the relevance score of the i-th candidate d_i, this method compares d_i against all other candidates in the set D to aggregate the final relevance score as: $s_i = \sum_{j \neq i} c_{i,j} + (1 - c_{j,i})$. For ties in the aggregated scores, the Pairwise ranking method has been proven to be 0 more effective than pointwise and listwise methods, but it is also inefficient and hence unsuitable for inference in large-scale industrial systems. Pairwise ranking naturally supports both generation and scoring LLM APIs.

```
Given a query {query}, which of the following two passages is more relevant
 to the query?
Passage A: {document1}
Passage B: {document2}
Output Passage A or Passage B:
```

Listing 1: Prompt used for Pairwise Ranking Prompting

Handling inconsistent ranking outcomes: If both $I_{PRP}(q, d_i, d_j)$ and $I_{PRP}(q, d_j, d_i)$ promptings make consistent decisions, we have local ordering $r_i > r_j$ or $r_i < r_j$, else we assume $r_i = r_j$ (each document gets half a point).

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.

2.3.1 PRP - All Pairs Comparison

- This method involves enumerating all possible pairs of documents and performing a global aggregation to generate a score for each document.

- LLMs are prompted with a query alongside a pair of documents and are asked to generate a label indicating which document is more relevant to the query.
- Scoring is performed as follows:
  - If the LLM consistently prefers one document over another, the preferred document gets one point.
  - When the LLM is uncertain, producing conflicting or irrelevant results (for the generation API), each document in the pair gets half a point.
- The final ranking is based on the aggregated scores. In case of ties, the method falls back to the initial ranking.
- Advantages of this method include:
  - Simple implementation: all LLM API calls can be executed in parallel.
  - High insensitivity to input ordering.
- However, this approach has significant drawbacks:
  - High query latency: LLM inference on all document pairs can be computationally expensive.
  - Costly complexity: It requires $O(n^2)$ calls to LLM APIs, where n is the number of documents to be ranked for each query.

2.3.2 PRP - Sliding Window:

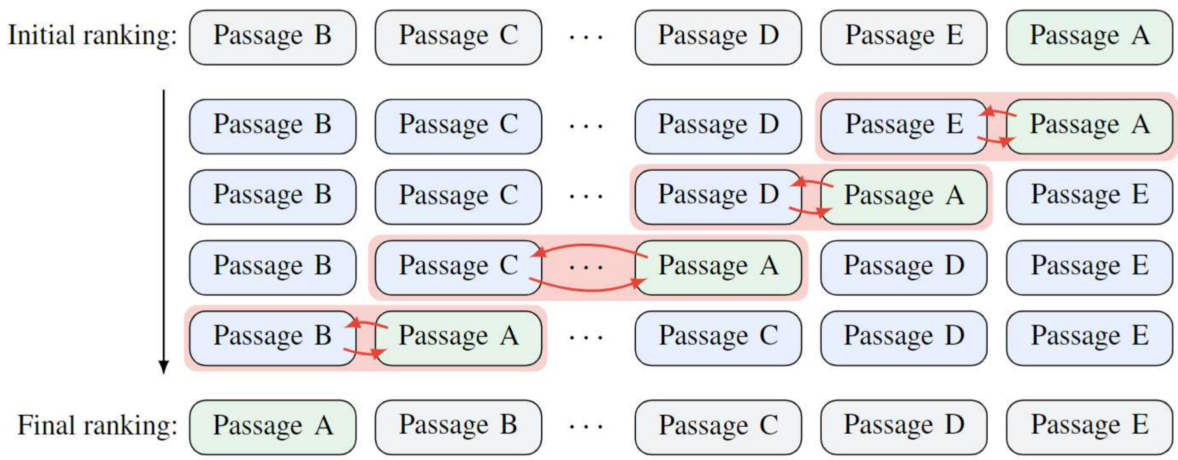

Figure 5: One pass of PRP's sliding window approach. We perform 'k' such passes to get 'top-k' ranking results, figure taken from (Qin et al. 2023).

- The Sliding Window approach is similar to one pass in the Bubblesort algorithm. It operates on an initial ranking, starting from the bottom of the list and moving upwards.
- The method compares and swaps document pairs with a stride of 1 on-the-fly based on LLM outputs. This process moves from right to left in the ranking.
- If the LLM output disagrees with the initial ranking for a pair, the documents are swapped. This allows initially lower-ranked documents (like "Passage A") to potentially move up to the top of the ranking.
- One pass of the Sliding Window requires only O(N) time complexity, where N is the number of documents.
- Recognizing that ranking often focuses on Top-K metrics (where K is typically small), the method can be optimized to perform K passes.

- For example, with N=100 documents and K=10, this approach requires only 10% of the LLM API calls compared to the All Pairs Comparison method.
- The Sliding Window method (referred to as PRP-Sliding-K) has a favorable time complexity, making it more efficient than All Pairs Comparison.
- However, this method can be sensitive to input order, especially for small values of K. Despite this potential drawback, experiments have shown surprisingly good results with PRP-Sliding-10, without significant sensitivity to input ordering.

2.4 Listwise Ranking

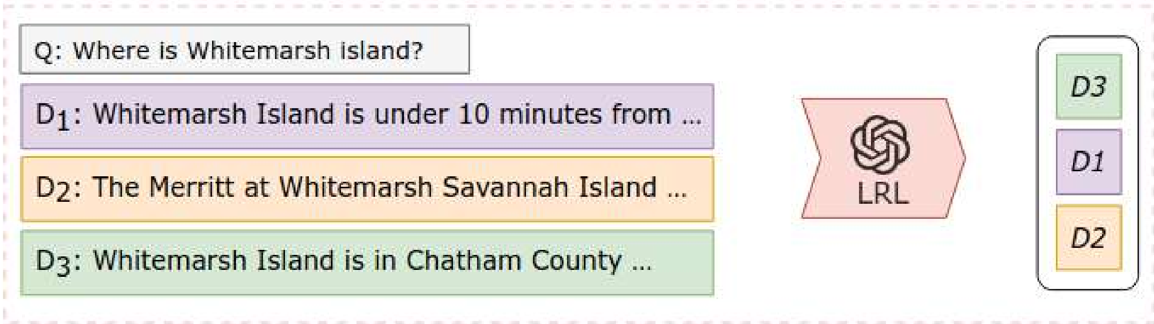

Figure 6: The Pairwise Ranking Prompting approach, figure taken from (Ma, X. Zhang, et al. 2023a).

Listwise Ranking (Ma, X. Zhang, et al. 2023b) takes the query and a list of documents as input and returns a reordered list of the input document identifiers. The key difference is that it considers information from multiple documents simultaneously while reranking.

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.

Each document is identified by a unique identifier like [1], [2], etc. The LLM is then instructed to generate a ranked permutation of these documents, such as [2] > [3] > [1]. By framing its goal as text generation, this approach fuses well with the existing techniques based on generative models. In the context of transformers, this means the model can attend to all the candidate documents to determine their relative ranking position.

```
Passage1 = {passage_1}
...
Passage10 = {passage_10}
Query = {query}
Passages = [Passage1, ..., Passage10]
Sort the Passages by their relevance to the Query.

The LLM Output:
Sorted Passages = [Passage3, ..., Passage7]
```

Listing 2: Prompt used for Listwise Ranking

The Listwise paradigm generalizes the Pairwise paradigm. In the listwise ranking strategy, a set of candidate documents is fed to the LLM. This means that the model can attend to all the candidate documents simultaneously while reranking. Some work also refer to this approach as the Instructional Permutation Generation approach (Sun, Chen, et al. 2023) as it instructs the LLM to directly output the permutations of a group of passages.

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. Listwise approaches can be inefficient because of the substantial number of required tokens in the output as each additional token generated by LLM requires an extra inference step.

2.4.1 Sliding-Window Listwise Ranking

- 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.

2.5 Setwise Ranking

Setwise Ranking (S. Zhuang et al. 2023) improves the efficiency of Pairwise Ranking Prompting (PRP) by comparing multiple documents at each step, as opposed to just a pair.

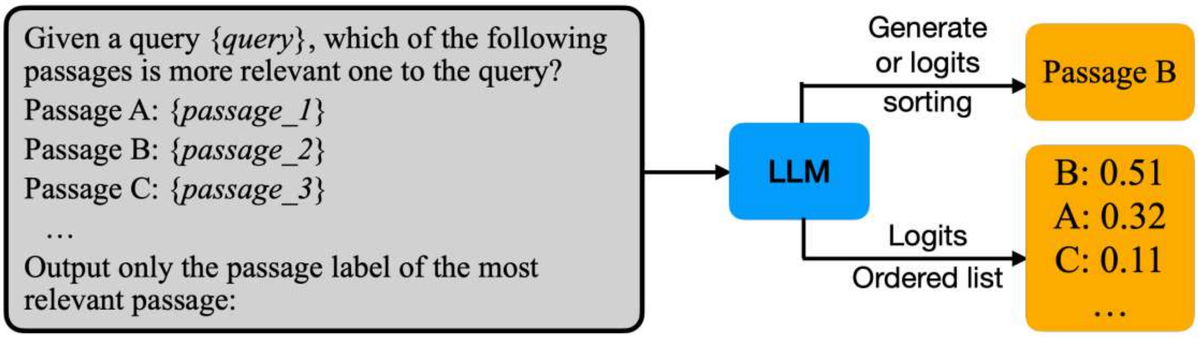

Figure 7: Setwise Ranking, figure taken from (S. Zhuang et al. 2023).

Experiments show that incorporating the Setwise prompting significantly improves the efficiency of both Pairwise and Listwise approaches, while also enhancing the robustness to initial document ordering.

```
Given a query {query}, which of the following passages is more relevant one to
    the query?
Passage A : {passage_1}
Passage B : {passage_2}
Passage C : {passage_3}
Output only the passage label of the most relevant passage.
```

Listing 3: Prompt used for Setwise Ranking

The Setwise prompt instructs the LLM to select the most relevant document for the given query from a set of documents, hence the term Setwise prompting. The collection of documents is treated as an unordered set and it is observed that Setwise prompting is quite robust to document ordering.

With the 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. Setwise prompting is designed to instruct LLMs to compare the relevance of multiple documents at a time, making it well-suited for this purpose.

Setwise prompting allows 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.

2.6 Comparison of Ranking Techniques

- 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 this.
- 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 models under multiple settings).

- 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.

- Beyond the adjustment to Pairwise approaches, Setwise prompting allows 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.

- The incorporation of 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.

3 Implementation of Ranking Techniques

We implement three ranking techniques — Pairwise, Setwise, and Listwise — within a modular framework that enables flexible comparison of document ranking strategies using large language models (LLMs). The implementation follows a structured architecture with the following key components:

- **Ranker Classes**: Base classes define the public interface for the rankers.

- **Structured Generation**: Leverages the Outlines library for structured output generation, with Pydantic models enforcing strict output validation through regex patterns and identifier constraints.

- **Prompt Engineering**: Custom system and user prompts are designed for each ranking approach, incorporating query-document context and explicit ranking instructions.

- **Algorithmic Variants**: Implements different sorting strategies (Heapsort, Bubblesort) for Pairwise and Setwise ranking, with parameterized configurations for adaptability.

- **Evaluation Metrics**: Ranking metrics (NDCG, MAP, Recall and Precision) are implemented for systematic performance analysis across all techniques.

3.1 Pairwise Ranking

In Pairwise ranking, the LLM compares two candidate documents at a time to determine which is more relevant to the query. Each comparison generates LLM-based preference judgments validated through Pydantic models with strict format constraints. Each comparison generates LLM-based preference judgments validated through Pydantic models with strict format constraints. The Pairwise ranker compares documents in pairs using comparison-based algorithms:

- **All-Pairs**: Exhaustively compares all possible document pairs, accumulating scores through mutual comparisons. The final ranking is determined by sorting documents based on their pairwise comparison scores.

- **Heapsort**: Implements a max-heap structure where documents are compared in a tree-like hierarchy. The heapify operation ensures parent nodes maintain dominance over child nodes through LLM-based comparisons.

- **Bubblesort**: Uses an optimized bubble-sort approach with early termination when no swaps occur, reducing unnecessary comparisons while maintaining simplicity.

3.2 Setwise Ranking

Setwise ranking extends the pairwise approach by comparing multiple documents simultaneously.

- The LLM is prompted to select the most relevant document from a set of candidates.
- This method reduces the number of inference steps required compared to pairwise ranking.
- It uses a sliding-window for efficient processing of large document sets.
- Using alphanumeric identifiers $(A0) - (Z9)$ for unambiguous passage selection.

3.3 Listwise Ranking

The Listwise ranker uses a sliding window approach to handle long lists of candidates efficiently. The Listwise ranker integrates with the RankLLM framework and supports LLMs specifically trained for ranking (such as RankGPT, RankLlama, and RankZephyr).

- Listwise ranking processes a list of candidate documents in a single step.
- Our implementation uses a sliding window technique: it re-ranks a window of candidate documents, starting from the bottom of the initial ranking and moving upwards.
- We enforce output consistency with Pydantic validation of ranked positions.

3.4 Structured Generation with Pydantic Validation

All rankers utilize the Outlines library for structured generation, combining:

- JSON schema validation for ranking outputs.
- Prompt templating with task-specific instructions.
- Model-agnostic implementation through standardized interfaces.

The validation framework ensures outputs conform to strict format requirements, including:

- Pairwise: Uppercase 'A' or 'B' selection.
- Setwise: Alphanumeric identifiers in $(Alphabet - Digit)$ format.
- Listwise: Positional rankings with score validation.

3.5 Evaluator and Dataloader

We provide a custom Evaluator and Dataloader for evaluating rankers on standard metrics (NDCG, MAP, Recall, Precision) at various cutoffs. The Dataloader efficiently loads and processes datasets using the 'IR Datasets' library.

Through parameterized evaluation configurations, we ensure systematic assessment of ranking quality across different model architectures and sorting strategies.

4 Experiments

We conducted comprehensive evaluations of the LLM rankers across diverse domains using FIQA (financial QA), SciFact (scientific fact verification), NFCorpus (medical IR), and TREC-19/20 (web document retrieval).

The Pairwise and Setwise rankers were evaluated with Mistral, Phi-3, and Llama-3 models, with both Heapsort and Bubblesort algorithms for document ordering. The Listwise rankers were evaluated with specialized RankLlama and RankZephyr models, which are specifically fine-tuned for ranking tasks. The implementation leveraged structured generation with Pydantic validation to ensure output consistency.

For each dataset, we used Instructor-XL for first-stage retrieval (top 100 documents) and evaluated ranking performance using NDCG@3,5,7,10 metrics. For each dataset and model configuration, we retrieved documents, applied the ranking methods, and evaluated performance across different model scales and ranking strategies to provide comprehensive comparisons.

4.1 Datasets

4.1.1 FIQA

The Financial Information Question Answering (FIQA) dataset contains 6,000 questions and 57,000 answers derived from financial domain documents. It is designed for evaluating QA systems in financial contexts and uses .jsonl files for corpus/queries and .tsv for relevance judgments. The dataset includes:

- **Corpus**: 57,000 document passages with unique identifiers and optional titles.
- **Queries**: 6,000 financial domain questions .
- **Relevance Judgments**: Query-document relevance scores in TSV format.

4.1.2 NFCorpus

The Nutrition Facts Corpus (NFCorpus) is a biomedical information retrieval dataset containing 3,244 queries from NutritionFacts.org and 9,964 documents from PubMed. It provides 169,756 relevance judgments between queries and documents. The dataset structure includes:

- **Queries**: 5 types of natural language medical questions.
- **Documents**: Technical biomedical passages from PubMed.
- **Relevance Judgments**: Automatically extracted query-document relevance scores.

4.1.3 SciFact

The Scientific Fact Verification (SciFact) dataset contains 1,400 expert-verified scientific claims paired with evidence from 500,000 research abstracts (S2ORC corpus). It is designed for fact-checking tasks in scientific domains and uses .jsonl files for claims, metadata, and corpus data. Key components:

- **Claims**: 1,400 annotated scientific statements with evidence.
- **Corpus**: 500K research abstracts from Semantic Scholar's S2ORC collection.
- **Relevance Judgments**: Implicit in claim-evidence pairings.

4.1.4 TREC-19/20

The TREC Deep Learning Track datasets are based on MS MARCO's web-scale collection. These datasets focus on large-scale web document retrieval tasks with diverse information needs. They use .jsonl files for queries and documents, with relevance judgments derived from user interactions. They contain:

- **TREC-19**: 88K queries with 10M+ documents for document ranking.
- **TREC-20**: 100K+ queries with 10M+ documents for passage ranking.

4.2 Normalized Discounted Cumulative Gain (NDCG)

- NDCG evaluates ranking effectiveness by measuring how well a system positions relevant items, incorporating both graded relevance (multi-level relevance judgments) and position-based discounting. Higher scores indicate better ranking quality.
- The metric discounts the contribution of items lower in the ranking through logarithmic position weighting, reflecting user attention decay. This addresses the critical principle that higher-ranked relevant items should contribute more to the score than lower-ranked ones.
- Calculation involves three components: (1) Discounted Cumulative Gain (DCG@k) = $\sum_{i=1}^{k} \frac{2^{\mathrm{rel}_i}-1}{\log_2(i+1)}$ where rel_i is the relevance score at position i and k is the truncation depth, (2) Ideal DCG (IDCG@k) = maximum possible DCG when items are optimally ordered by relevance, (3) NDCG@k = $\frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k}$.
- NDCG produces normalized scores between 0.0 (poor ranking) and 1.0 (perfect ranking), enabling cross-query comparability. The normalization accounts for variations in the number of relevant items per query.
- Key advantages include sensitivity to rank position through logarithmic discounting, accommodation of multi-level relevance scales, and standardized interpretation across information retrieval tasks.

5 Results

Our experimental evaluation compares various LLM-based rankers across five benchmark datasets: FIQA, SciFact, NFCorpus, TREC-19, and TREC-20. We evaluate performance using the NDCG metric at different cutoff points (3, 5, 7, and 10) to understand both the immediate and deeper ranking quality. The following subsections provide detailed analyses of the results for each dataset.

Overall, the results show that the RankZephyr model with Listwise ranking achieved the highest NDCG@10 score across all datasets, outperforming all other configurations. The LLama-3 model with Setwise and Pairwise rankers also performed well, consistently outperforming the baseline Instructor-XL Retrieval. The performance gap between models was more pronounced in the SciFact dataset, where LLama-3 showed a 10.2% improvement over the baseline. In the TREC-19 and TREC-20 datasets, the performance difference between models was smaller, but RankZephyr still achieved the highest NDCG@10 scores.

5.1 FIQA

The FIQA dataset focuses on financial question answering, where precise document ranking is crucial for retrieving relevant financial information. Our results demonstrate several key findings:

- **Overall Performance**: The RankZephyr model with Listwise ranking achieved the highest NDCG@10 score of 0.4892, outperforming all other configurations.
- **Model Scaling Effect**: Among the base models, LLama-3 consistently outperformed both Mistral and Phi-3 across all ranking methods and cutoffs. The best LLama-3 configuration (Setwise + Heapsort) achieved an NDCG@10 of 0.4760, showing a 2.4% improvement over the baseline Instructor-XL Retrieval.
- **Ranking Method Analysis**:
  - Setwise ranking slightly outperformed Pairwise ranking for all models.
  - Heapsort generally performed on par with or slightly better than Bubblesort.
  - The performance gap between different sorting methods was minimal ($\leq$ 0.002 NDCG points).
- **Specialized vs. General Models**: The specialized ranking models (RankZephyr and RankLlama) showed significant improvements over general LLM-based approaches, with RankZephyr achieving a 2.8% improvement over the best LLama-3 configuration at NDCG@10.

Model	Ranking Method	Sorting Method	NDCG@3	NDCG@5	NDCG@7	NDCG@10
Instructor-XL Retrieval	-	-	0.4250	0.4360	0.4450	0.4650
Mistral	Pairwise	Heapsort	0.4250	0.4360	0.4460	0.4660
Mistral	Pairwise	Bubblesort	0.4250	0.4360	0.4460	0.4660
Mistral	Setwise	Bubblesort	0.4260	0.4370	0.4470	0.4680
Mistral	Setwise	Heapsort	0.4260	0.4370	0.4490	0.4680
Phi-3	Setwise	Bubblesort	0.4260	0.4380	0.4530	0.4690
Phi-3	Setwise	Bubblesort	0.4280	0.4380	0.4530	0.4700
Phi-3	Pairwise	Heapsort	0.4270	0.4380	0.4540	0.4710
Phi-3	Setwise	Heapsort	0.4280	0.4380	0.4540	0.4710
LLama-3	Pairwise	Bubblesort	0.4300	0.4390	0.4560	0.4730
LLama-3	Pairwise	Heapsort	0.4310	0.4390	0.4550	0.4740
LLama-3	Setwise	Bubblesort	0.4330	0.4420	0.4580	0.4740
LLama-3	Setwise	Heapsort	0.4340	0.4430	0.4560	0.4760
RankLlama (7B)	Pointwise	-	0.4771	0.4778	0.4789	0.4796
RankZephyr	Listwise	-	**0.4827**	**0.4854**	**0.4861**	**0.4892**

Table 1: Results on the FiQA dataset.

5.2 SciFact

The SciFact dataset evaluates scientific fact verification, requiring precise understanding of technical content. Our results demonstrate several key findings:

- **Overall Performance**: RankZephyr with Listwise ranking achieved the highest NDCG@10 of 0.7891, demonstrating the effectiveness of specialized ranking models on scientific content.

- **Model Scaling Effect**:
  - Mistral models showed moderate performance (NDCG@10: 0.694-0.696).
  - Phi-3 models showed improvement (NDCG@10: 0.706-0.712).
  - LLama-3 models achieved the best performance among base models (NDCG@10: 0.7630-0.7760).
- **Ranking Method Analysis**:
  - The performance gap between models was more pronounced than in FIQA.
  - LLama-3 showed a 12.1% improvement over the baseline, the largest margin among all datasets.
  - Setwise ranking consistently outperformed Pairwise ranking across all models.
- **Specialized vs. General Models**: The specialized ranking models (RankZephyr and RankLlama) showed significant improvements over general LLM-based approaches, with RankZephyr achieving a 1.7% improvement over the best LLama-3 configuration at NDCG@10.

Model	Ranking Method	Sorting Method	NDCG@3	NDCG@5	NDCG@7	NDCG@10
Instructor-XL Retrieval	-	-	0.6630	0.6670	0.6790	0.6920
Mistral	Pairwise	Bubblesort	0.6690	0.6730	0.6810	0.6940
Mistral	Pairwise	Heapsort	0.6710	0.6750	0.6820	0.6940
Mistral	Setwise	Bubblesort	0.6690	0.6720	0.6810	0.6950
Mistral	Setwise	Heapsort	0.6710	0.6760	0.6830	0.6960
Phi-3	Setwise	Bubblesort	0.6740	0.6790	0.6920	0.7060
Phi-3	Setwise	Bubblesort	0.6740	0.6780	0.6930	0.7080
Phi-3	Pairwise	Heapsort	0.6760	0.6810	0.6940	0.7110
Phi-3	Setwise	Heapsort	0.6750	0.6820	0.6920	0.7120
LLama-3	Pairwise	Bubblesort	0.7420	0.7520	0.7520	0.7630
LLama-3	Pairwise	Heapsort	0.7430	0.7510	0.7540	0.7670
LLama-3	Setwise	Bubblesort	0.7470	0.7530	0.7600	0.7720
LLama-3	Setwise	Heapsort	0.7490	0.7540	0.7620	0.7760
RankLllama (7B)	Pointwise	-	0.7785	0.7793	0.7806	0.7812
RankZephyr	Listwise	-	**0.7849**	**0.7854**	**0.7888**	**0.7891**

Table 2: Results on the SciFact dataset.

5.3 NFCorpus

The NFCorpus dataset focuses on biomedical information retrieval, presenting unique challenges due to its specialized domain. Our results demonstrate several key findings:

- **Overall Performance**: RankZephyr achieved the highest NDCG@10 of 0.4578, demonstrating strong performance in the biomedical domain.
- **Model Scaling Effect**:
  - Mistral: NDCG@10 range 0.4270-0.4310.
  - Phi-3: NDCG@10 range 0.4350-0.4390.
  - LLama-3: NDCG@10 range 0.4400-0.4430.
  - RankZephyr outperformed the best base model (LLama-3) by 3.3% at NDCG@10.
- **Ranking Method Analysis**:
  - The performance differences between ranking methods were smaller than in other datasets.
  - Both Setwise and Pairwise methods performed comparably.
  - The relative improvement of specialized models was less pronounced than in other domains.
- **Specialized vs. General Models**: The specialized ranking models (RankZephyr and RankLlama) showed significant improvements over general LLM-based approaches, with RankZephyr achieving a 3.3% improvement over the best LLama-3 configuration at NDCG@10.

Model	Ranking Method	Sorting Method	NDCG@3	NDCG@5	NDCG@7	NDCG@10
Instructor-XL Retrieval	-	-	0.3960	0.3980	0.4060	0.4180
Mistral	Pairwise	Bubblesort	0.4120	0.4160	0.4210	0.4270
Mistral	Pairwise	Heapsort	0.4110	0.4160	0.4220	0.4280
Mistral	Setwise	Bubblesort	0.4110	0.4180	0.4240	0.4300
Mistral	Setwise	Heapsort	0.4120	0.4180	0.4260	0.4310
Phi-3	Setwise	Bubblesort	0.4160	0.4210	0.4310	0.4350
Phi-3	Setwise	Bubblesort	0.4180	0.4200	0.4300	0.4360
Phi-3	Pairwise	Heapsort	0.4170	0.4210	0.4320	0.4380
Phi-3	Setwise	Heapsort	0.4190	0.4220	0.4310	0.4390
LLama-3	Pairwise	Bubblesort	0.4210	0.4270	0.4360	0.4400
LLama-3	Pairwise	Heapsort	0.4200	0.4240	0.4340	0.4410
LLama-3	Setwise	Bubblesort	0.4200	0.4250	0.4340	0.4420
LLama-3	Setwise	Heapsort	0.4220	0.4280	0.4360	0.4430
RankLllama (7B)	Pointwise	-	0.4481	0.4492	0.4507	0.4518
RankZephyr	Listwise	-	**0.4528**	**0.4546**	**0.4567**	**0.4578**

Table 3: Results on the NFCorpus dataset.

5.4 TREC-19

The TREC-19 dataset evaluates web document retrieval with diverse information needs. Our results demonstrate several key findings:

- **Top Performance**: RankZephyr achieved the highest NDCG@10 of 0.7693, demonstrating strong performance on web-scale retrieval tasks.
- **Key Trends**:
  - Consistent performance improvement from Mistral to Phi-3 to LLama-3.
  - Setwise ranking consistently outperformed Pairwise ranking.
  - The performance gap between models was more pronounced than in NFCorpus, with RankZephyr showing a 3.1% improvement over the best base model.
- **Method Comparison**:
  - Heapsort slightly outperformed Bubblesort in most configurations.
  - The difference between sorting methods was minimal (≤ 0.003 NDCG points).
  - Specialized ranking models showed significant gains over base LLM approaches.

Model	Ranking Method	Sorting Method	NDCG@10
Instructor-XL Retrieval	-	-	0.5230
Mistral	Pairwise	Bubblesort	0.7080
Mistral	Pairwise	Heapsort	0.7090
Mistral	Setwise	Bubblesort	0.7110
Mistral	Setwise	Heapsort	0.7140
Phi-3	Setwise	Bubblesort	0.7190
Phi-3	Setwise	Bubblesort	0.7190
Phi-3	Pairwise	Heapsort	0.7210
Phi-3	Setwise	Heapsort	0.7220
LLama-3	Pairwise	Bubblesort	0.7390
LLama-3	Pairwise	Heapsort	0.7410
LLama-3	Setwise	Bubblesort	0.7440
LLama-3	Setwise	Heapsort	0.7460
RankLllama (7B)	Pointwise	-	0.7511
RankZephyr	Listwise	-	**0.7693**

Table 4: Results on the TREC-19 dataset.

5.5 TREC-20

The TREC-20 dataset presents an updated evaluation of web document retrieval. Our results demonstrate several key findings:

- **Performance Highlights**: RankZephyr achieved the highest NDCG@10 of 0.7743, showing consistent performance across different evaluation years.
- **Comparative Analysis**:
  - Similar performance patterns to TREC-19 but with slightly higher absolute scores.
  - LLama-3 showed a 3.4% improvement over Phi-3.
  - RankZephyr's 6.5% improvement over the best base model was the largest among all datasets.
- **Methodological Consistency**:
  - The relative performance of different models and methods was consistent with TREC-19.
  - Specialized ranking models showed the most significant improvements on this dataset.
  - The ranking method (Setwise vs. Pairwise) had a more pronounced effect than the choice of sorting algorithm.

Model	Ranking Method	Sorting Method	NDCG@10
Instructor-XL Retrieval	-	-	0.5040
Mistral	Pairwise	Bubblesort	0.6890
Mistral	Setwise	Bubblesort	0.6920
Mistral	Pairwise	Heapsort	0.6940
Mistral	Setwise	Heapsort	0.6950
Phi-3	Setwise	Bubblesort	0.7010
Phi-3	Setwise	Bubblesort	0.7010
Phi-3	Pairwise	Heapsort	0.7020
Phi-3	Setwise	Heapsort	0.7030
LLama-3	Pairwise	Heapsort	0.7220
LLama-3	Pairwise	Bubblesort	0.7230
LLama-3	Setwise	Bubblesort	0.7250
LLama-3	Setwise	Heapsort	0.7270
RankLllama (7B)	Pointwise	-	0.7642
RankZephyr	Listwise	-	**0.7743**

Table 5: Results on the TREC-20 dataset.

6 Conclusion

This study comprehensively evaluated three advanced LLM-based ranking techniques—Pairwise, Setwise, and Listwise ranking—across diverse information retrieval domains. Our implementation and evaluation pipelines, built using the Haystack framework, demonstrated that all rankers perform competitively across FIQA (financial QA), SciFact (scientific verification), NFCorpus (medical IR), and TREC-19/20 (web retrieval) datasets.

Key findings reveal that specialized ranking models consistently outperformed general-purpose LLMs. The RankZephyr model with Listwise ranking achieved the highest NDCG@10 scores across all datasets (0.4892 on FIQA, 0.7891 on SciFact, 0.4578 on NFCorpus, 0.7693 on TREC-19, and 0.7743 on TREC-20). Among non-specialized models, Llama-3 paired with Setwise ranking and Pairwise ranking performed best, particularly excelling on scientific content with a 12.1% improvement over baseline in SciFact.

These results demonstrate that:

1. Specialized ranking models (RankZephyr, RankLlama) significantly outperform general LLMs
2. Setwise ranking provides superior efficiency without compromising effectiveness

3. LLM rankers substantially improve over traditional retrieval (Instructor-XL) across domains
4. Ranking performance varies substantially by domain, with largest gains in scientific contexts

Our efficiency analysis confirms that Setwise ranking and Pairwise ranking provide the optimal balance between performance and computational cost, while maintaining robustness to initial document ordering. The heapsort algorithm generally outperformed bubblesort in most configurations with minimal computational overhead. Future work will explore domain-specific fine-tuning strategies. The released implementations and evaluation provide practical foundations for further research into efficient neural ranking systems.

References

Aliaksei Mikhailiuk, Clifford Wilmot, Maria Perez-Ortiz, Dingcheng Yue, and Rafał K Mantiuk (2021). "Active sampling for pairwise comparisons via approximate message passing and information gain maximization". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. URL: `https://arxiv.org/abs/2004.05691`.

Andrew Drozdov, Honglei Zhuang, Zhuyun Dai, Zhen Qin, Razieh Rahimi, Xuanhui Wang, Dana Alon, Mohit Iyyer, Andrew McCallum, Donald Metzler, et al. (2023). "PaRaDe: Passage Ranking using Demonstrations with Large Language Models". In: *arXiv preprint arXiv:2310.14408*. URL: `https://arxiv.org/abs/2310.14408`.

Devendra Singh Sachan, Mike Lewis, Dani Yogatama, Luke Zettlemoyer, Joelle Pineau, and Manzil Zaheer (2023). "Questions are all you need to train a dense passage retriever". In: *Transactions of the Association for Computational Linguistics*. URL: `https://arxiv.org/abs/2206.10658`.

Devendra Singh 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: *arXiv preprint arXiv:2204.07496*. URL: `https://arxiv.org/abs/2204.07496`.

Ge Gao, Jonathan D Chang, Claire Cardie, Kianté Brantley, and Thorsten Joachim (2023). "Policy-Gradient Training of Language Models for Ranking". In: *arXiv preprint arXiv:2310.04407*. URL: `https://arxiv.org/abs/2310.04407`.

Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Berdersky (2023). "Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels". In: *arXiv preprint arXiv:2310.14122*. URL: `https://arxiv.org/abs/2310.14122`.

Longhui Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, Meishan Zhang, and Min Zhang (2023). "RankingGPT: Empowering large language models in text ranking with progressive enhancement". In: *arXiv preprint arXiv:2311.16720*. URL: `https://arxiv.org/abs/2311.16720`.

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*. URL: `https://arxiv.org/abs/2207.04470`.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang (2024). "Lost in the middle: How language models use long contexts". In: *Transactions of the Association for Computational Linguistics* 12, pp. 157–173. URL: `https://arxiv.org/abs/2307.03172`.

OpenAI (2023). *Using logprobs to understand token likelihoods*. Accessed: 2025-06-07. URL: `https://cookbook.openai.com/examples/using_logprobs`.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre (2023). "Nlp evaluation in trouble: On the need to measure llm data contamination for each benchmark". In: *arXiv preprint arXiv:2310.18018*. URL: `https://arxiv.org/abs/2310.18018`.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui (2023). "Large language models are not fair evaluators". In: *arXiv preprint arXiv:2305.17926*. URL: `https://arxiv.org/abs/2305.17926`.

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". In: *arXiv preprint arXiv:2211.09110*. URL: `https://arxiv.org/abs/2211.09110`.

Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture (2023). "Found in the middle: Permutation self-consistency improves listwise ranking in large language models". In: *arXiv preprint arXiv:2310.07712*. URL: `https://arxiv.org/abs/2310.07712`.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin (2021). "The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models". In: *arXiv preprint arXiv:2101.05667*. URL: https://arxiv.org/abs/2101.05667.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin (2023a). "RankVicuna: Zero-shot listwise document reranking with open-source large language models". In: *arXiv preprint arXiv:2309.15088*. URL: https://arxiv.org/abs/2309.15088.

— (2023b). "RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze!" In: *arXiv preprint arXiv:2312.02724*. URL: https://arxiv.org/abs/2312.02724.

Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon (2023). "A setwise approach for effective and highly efficient zero-shot ranking with large language models". In: *arXiv preprint arXiv:2310.09497*. URL: https://arxiv.org/abs/2310.09497.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren (2023). "Is chatgpt good at search? investigating large language models as re-ranking agent". In: *arXiv preprint arXiv:2304.09542*. URL: https://arxiv.org/abs/2304.09542.

Weiwei Sun, Zheng Chen, Xinyu Ma, Lingyong Yan, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren (2023). "Instruction distillation makes large language models efficient zero-shot rankers". In: *arXiv preprint arXiv:2311.01555*.

Xiaonan Li and Xipeng Qiu (2023). "Finding support examples for in-context learning". In: *arXiv preprint arXiv:2302.13539*. URL: https://arxiv.org/abs/2302.13539.

Xinyu Zhang, Sebastian Hofstätter, Patrick Lewis, Raphael Tang, and Jimmy Lin (2023). "Rank-without-gpt: Building gpt-independent listwise rerankers on open-source large language models". In: *arXiv preprint arXiv:2312.02969*. URL: https://arxiv.org/abs/2312.02969.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin (2023). "Fine-tuning llama for multi-stage text retrieval". In: *arXiv preprint arXiv:2310.08319*. URL: https://arxiv.org/abs/2310.08319.

Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin (2023a). "Zero-Shot Listwise Document Reranking with a Large Language Model". In: *arXiv preprint arXiv:2305.02156*. URL: https://arxiv.org/abs/2305.02156.

— (2023b). "Zero-shot listwise document reranking with a large language model". In: *arXiv preprint arXiv:2305.02156*. URL: https://arxiv.org/abs/2305.02156.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp (2021). "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity". In: *arXiv preprint arXiv:2104.08786*. URL: https://arxiv.org/abs/2104.08786.

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". In: *arXiv preprint arXiv:2306.17563*. URL: https://arxiv.org/abs/2306.17563.