LongRanker: Efficient One-Pass Document Reranking with Long-Context Large Language Models

Changjiang Zhou
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
zhouchangjiang23s@ict.ac.cn

Ruqing Zhang*
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
zhangruqing@ict.ac.cn

Jiafeng Guo*
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
guojiafeng@ict.ac.cn

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

Yixing Fan
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
fanyixing@ict.ac.cn

Xueqi Cheng
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
cxq@ict.ac.cn

Abstract

Large language models (LLMs) have demonstrated significant potential in listwise document reranking. Due to their limited context length, LLM-based listwise reranking methods often rely on a sliding window strategy that only processes a small subset of documents at a time. While effective, this approach lacks interactions between documents, increases computational overhead, and results in significant API costs. It is crucial to develop long-context LLMs for enabling the full ranking of all documents in one pass.

We propose a long-context listwise document reranker, LongRanker, and make two major contributions to enable long-context LLMs for listwise reranking: (i) To improve length extrapolation for listwise inputs, we introduce an *intra-inter hierarchical positional encoding* approach that combines intra-document encoding to identify token locations within a document with inter-document encoding to specify the document index. (ii) To efficiently model long contexts with reduced computational costs, we introduce a *top-k attention pruning* mechanism; it ensures that the model attends to the query and the most important documents while disregarding less relevant ones. Extensive empirical results demonstrate that LongRanker achieves performance comparable to state-of-the-art rerankers while significantly improving efficiency through reduced LLM calls.

CCS Concepts

• **Information systems → Retrieval models and ranking**.

Keywords

Listwise reranking, Length extrapolation, Attention pruning

ACM Reference Format:
Changjiang Zhou, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Yixing Fan, and Xueqi Cheng. 2026. LongRanker: Efficient One-Pass Document Reranking with Long-Context Large Language Models. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3774904.3792179

*Jiafeng Guo and Ruqing Zhang are the corresponding authors.

1 Introduction

Large language models (LLMs) have transformed various fields with their zero-shot and few-shot capabilities in language understanding and reasoning. Recent research has explored the application of LLMs to listwise document reranking in the field of information retrieval (IR) [30, 33, 36, 43, 48, 49]. The key idea is to instruct LLMs to generate a ranked permutation of document identifiers based on the relevance to a given query of its associated candidate document list. By exploiting their ability to use semantic context for inter-document comparisons and interactions, listwise rerankers have achieved state-of-the-art performance across multiple IR datasets [7, 26, 36, 49].

Context length limitations. Due to the limited context length of many LLMs, existing listwise reranking methods, e.g., RankGPT [36] and RankVicuna [25], often employ a heuristic *sliding window strategy* to process the entire document list. This strategy processes only a subset of documents at a time and iteratively reranks these subsets in a back-to-first order. By using fixed window and step sizes, the strategy allows relevant documents initially ranked low to be promoted to higher positions. The sliding window strategy has demonstrated empirical effectiveness, but it also has several key limitations: (i) LLMs can only assess the local inter-document context within the sliding window at a given time. This prevents the model from using the global context across all candidate documents, potentially hindering reranking performance. (ii) The sliding window strategy increases the number of LLM calls, leading to higher computational complexity, longer response time latency, and greater economic costs. (iii) Due to overlapping regions between adjacent windows, many documents are evaluated multiple times, resulting in significant redundancy and possible inconsistencies.

One promising solution is to extend the context window size of the LLM. Recent proposed long-context techniques [2, 22, 46] usually optimize positional encoding at the token level to support longer input lengths. However, for listwise reranking, where the goal is to generate a ranked permutation of documents, it is essential to extend the context window from the *token level* to the *document level* to ensure the effectiveness of the resulting permutation. Designing context window extensions specifically tailored for listwise reranking remains a challenging and unresolved task.

A long-context listwise document reranker. Our goal is to develop an effective and efficient listwise document reranker using LLMs, named LongRanker, capable of accommodating all candidate documents for reranking while maintaining relatively low computational overhead. To achieve this, we address two key challenges.

First, *how to improve length extrapolation for listwise document input?* Long listwise document inputs may result in an out-of-distribution (OOD) issue related to positional encoding as LLMs are typically not exposed to such lengthy inputs during pre-training. To address this, we propose a *hierarchical positional encoding* for listwise reranking, which incorporates two encodings for each position: an *intra-document encoding* to identify the token's location within its document and an *inter-document encoding* to indicate the document it belongs to. (i) For intra-document encoding, we use the original absolute positional encoding in the transformer [42] since the number of tokens within a document is limited. (ii) For inter-document positional encoding, we use a rotary position encoding [35] to specify the document and capture contextual relationships between documents. This hierarchical approach introduces inductive biases at two levels, each targeting distinct aspects of positional information. By effectively integrating these biases into model architectures, the learning process is improved, mitigating the OOD problem caused by long listwise document inputs.

Second, *how to efficiently handle long-context inputs with low computation costs?* Building upon the aforementioned hierarchical positional encoding, all candidate documents along with the user query can effectively be fed to the LLMs simultaneously. However, long-context inputs introduce the challenge of computational explosion in attention mechanisms, leading to an increase in the overall LLM computational complexity [1, 6]. Therefore, we introduce the *top-k attention pruning*, aiming to attend to the most relevant documents while masking the remaining under the premise of attending to the beginning query. The core idea is inspired by the natural assumption in top-k ranking, where users tend to prioritize top-ranked objectives in real-world ranking applications [12]. Drawing inspiration from the hierarchical generative process of top-k ranking [18], our method has three key steps for constructing the attention mask: (i) Identify the top-k most relevant documents for fine-grained attention. (ii) Ensure tokens within each document attend to the user query at every step. (iii) Preserve inter-document attention within top-k documents while suppressing it for non-top-k documents. The top-k attention pruning module is highly flexible and can be seamlessly integrated with hierarchical positional encoding through chunk-aware attention masks during fine-tuning and inference stems.

Experimental findings. We adopt Qwen2.5-1.5B-Instruct [39, 47] with 1.5B parameters as the backbone architecture of LongRanker. We insert hierarchical positional encoding following the implementation of BiPE_RoPE [9] and implement the top-k attention pruning through chunk-aware attention masks. We conduct experiments on three widely used benchmark datasets, i.e., TREC-DL-19, TREC-DL-20 and BEIR [4, 40, 44]. Extensive evaluation shows that LongRanker can achieve impressive advantages in terms of effectiveness and efficiency. Compared to the state-of-the-art LLM-based supervised reranker RankMistral$_{100}$, LongRanker demonstrates enhancements of 1.08%, 0.89%, and 0.13% in nDCG@10 on the TREC-DL-19, TREC-DL-20, and BEIR datasets, respectively. Furthermore, LongRanker exhibits a substantial efficiency advantage, achieving significantly lower time latency.

2 Preliminaries

We first formalize the task of listwise document reranking and subsequently review rotary position embedding, a widely employed method of relative positional encoding.

2.1 Listwise document reranking with LLMs

The core idea of listwise rerankers is to prompt LLMs to generate permutations of document identifiers based on their relevance to the user query. A typical input prompt to LLMs is as follows:

```
I will provide you with {num} passages, each indicated
by a numerical identifier []. Rank the passages based
on their relevance to the search query: {query}.
[1] {document 1}
[2] {document 2}
...
[num] {document num}
Search query: {query}. Rank the {num} passages above
based on their relevance to the search query. All
the passages should be included and listed using
identifiers, in descending order of relevance. The
output format should be [] > [], e.g., [4] > [2].
Only respond with the ranking results, do not say any
word or explain.
```

By feeding the prompt into LLMs, they can be directed to generate a permutation of document identifiers, thereby yielding a ranked list that reflects the relevance of each document to the user query, with a standard output O adhering to the following format:

```
[4] > [2] > [1] > [3] > [···]
```

To this end, the input prompt sequence S can be formally defined as $S = [p, q, d_1, d_2, \dots, d_i, \dots, d_n]$, where n denotes the number of candidate documents. Herein, p represents the instruction prompt that guides LLMs to generate permutations of document identifiers, q denotes the user query, and each d_i refers to an individual document with a leading numeric identifier $[i]$, and we further denote d_i as $[w_i^1, w_i^2, \dots, w_i^j, \dots, w_i^{|d_i|}]$, where each w_i^j denotes the j-th token within d_i. Upon formalizing the listwise document input, it becomes evident that each token within a document possesses a bilevel hierarchical index, i.e., the document index i and the intra-document token index j. This observation inspires us to devise a unique model architecture tailored for listwise document reranking.

2.2 Rotary position embedding

Rotary position embedding (RoPE) [35] is a typical relative positional encoding employed within Transformer-based [42] LLMs such as LLaMA [41] and Mistral [10]. RoPE can introduce relative positional information through the implementation of absolute positional encoding. Given a d-dimension embedding vector $x = (x_0, x_1, \dots, x_{d-1})^\top$ and a position index m, the vector-valued complex function $f(x,m)$ defined by RoPE is as follows:

$$f(x,m) = \begin{pmatrix} (x_0 + ix_1)e^{im\theta_1} \\ (x_2 + ix_3)e^{im\theta_2} \\ \vdots \\ (x_{d-2} + ix_{d-1})e^{im\theta_{d/2}} \end{pmatrix}, \tag{1}$$

where $i = \sqrt{-1}$ is the imaginary unit, $\theta_j = b^{-\frac{2j}{d}}$ denotes the rotation angle corresponding to the dimension j, herein b denotes the base frequency of RoPE and is typically set to 10,000 for current LLMs. The f function is applied to both q and k. Theoretically, applying the f function to the vector x is equivalent to applying the rotation matrix $R_{m,\theta}$ to the vector x, which can be formulated as follows:

$$f(x,m) = R_{m,\theta}x. \tag{2}$$

The original attention mechanism in Transformer-based models can be defined as follows:

$$A_{ij} = q_i^\top k_j, \tag{3}$$

$$Attention(q,k,v) = \text{softmax}\left(\frac{A}{\sqrt{d}}\right)v, \tag{4}$$

where $A \in \mathbb{R}^{L\times L}$ denotes the attention score matrix, $q,k,v \in \mathbb{R}^d$ denotes the query, key and value embedding respectively. RoPE introduces positional information into each token by employing the f function to modify the original attention mechanism. Specifically, by means of applying the f function, the RoPE can leverage rotation matrices when calculating the attention scores via Equation 3 as:

$$A_{ij} = f(q_i,i)^\top f(k_j,j) = (R_{i,\theta}q_i)^\top (R_{j,\theta}k_j) \tag{5}$$

$$= q_i^\top R_{i,\theta}^\top R_{j,\theta}k_j = q_i^\top R_{j-i,\theta}k_j. \tag{6}$$

The RoPE method enables models to comprehend relative positions between tokens by incorporating positional information into the attention mechanism, thereby demonstrating the capacity to handle longer sequences that are unseen during training in some degree. Nevertheless, RoPE is still unable to effectively generalize to sequences that significantly exceed the length encountered during the pre-training phase. This limitation hinders its direct applicability in handling long-context inputs for listwise document reranking.

3 Our Method

Similar to previous LLMs [51], LongRanker employs a transformer-based model architecture while including two specialized components for long-context listwise document reranking: (i) Hierarchical positional encoding is designed to improve length extrapolation for listwise inputs. (ii) Top-k attention pruning is devised to handle long-context listwise inputs with low computation overhead.

3.1 Hierarchical positional encoding

As formulated in Section 2.1, each token within a document possesses a bilevel hierarchical index, i.e., the document index i and the intra-document token index j. The structural characteristic inspires us to propose a hierarchical positional encoding to improve length extrapolation for listwise document input. The key idea is to decompose the rapidly expanding original position index into the intra-document token index and the document index, and then model them with two components: intra-document positional encoding and inter-document positional encoding. Given that the number of intra-document tokens remains relatively consistent,

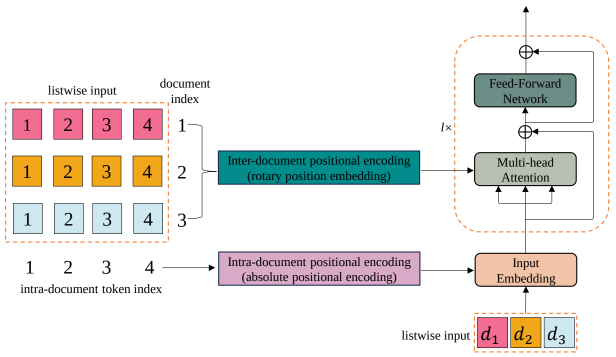

listwise input
document index
1
2
3
4
1
2
3
Inter-document positional encoding (rotary position embedding)
Intra-document positional encoding (absolute positional encoding)
intra-document token index
l×
Feed-Forward Network
Multi-head Attention
Input Embedding
listwise input
d_1
d_2
d_3

Figure 1: Hierarchical positional encoding. For simplicity, the instruction prompt and user query are omitted from the diagram. We assume there are three input documents and each document contains four tokens.

and the number of documents increases relatively slowly in the reranking task (typically not exceeding 1,000), this novel hierarchical positional encoding can mitigate the position OOD problem, thereby enhancing length extrapolation for listwise document input. Figure 1 visualizes the hierarchical positional encoding.

Intra-document positional encoding. As formulated in Section 2.1, each document $d_i = [w_i^1, w_i^2, \ldots, w_i^j, \ldots, w_i^{|d_i|}]$ forms an independent module in the listwise document input. The intra-document positional encoding is strategically designed to pinpoint the location of each token within its respective document, thereby augmenting the model's comprehension of the semantics inherent in each individual document. Given that the number of tokens within each document is constrained in listwise reranking, with the prevailing convention in existing studies typically truncating the length of each document $|d_i|$ to not exceed 100 [25, 26, 36], it is posited that the original absolute positional encoding [42] is sufficient to model the intra-document token index. Formally, for each token w_i^j, a real-valued position embedding pe^j is assigned:

$$pe^j_{2m} = \sin(j/10000^{2m/d_{hidden}}), \tag{7}$$

$$pe^j_{2m+1} = \cos(j/10000^{2m/d_{hidden}}), \tag{8}$$

where j is the position, m is the dimension pointer, and d_{hidden} denotes the dimension of hidden states. pe^j will be subsequently added to the token embedding of w_i^j, retaining the structural and contextual integrity within the document d_i. Notably, the intra-document positional encoding of w_i^j is independent of the document index i. This implies that the same intra-document positional encoding is consistently shared across different documents.

Inter-document positional encoding. We utilize RoPE mentioned in Section 2.2 to specify the document to which each token belongs. Formally, suppose the query and key embedding of the token w_i^j in the original multi-head attention module is q_i^j and k_i^j respectively, RoPE can be integrated by applying the f function mentioned in Equation 1:

$$q_i^j \leftarrow f(q_i^j, i), \quad k_i^j \leftarrow f(k_i^j, i). \tag{9}$$

Notably, each token w_i^j within the document d_i shares the same document index i.

By modeling the intrinsic structure of listwise document input using hierarchical positional encoding, LLMs are empowered to handle long-context listwise input while capturing inter-document boundaries. It's worth noting that the instruction prompt p and the user query q are modeled as a special type of document for the sake of simplicity and uniformity.

Discussion. He et al. [9] also devised a bilevel positional encoding dubbed BiPE. However, their approach diverges from ours in two primary respects: (i) Our method is inspired by the intrinsic structure of listwise document inputs, which inherently segment inputs into intra- and inter-document levels. In contrast, their strategy relies on symbol detection for segmentation (e.g., newline and full stop). (ii) Their approach focuses on lowering language modeling perplexity during the pre-training phase, whereas our method is specifically tailored for listwise document reranking and is implemented during the fine-tuning phase.

3.2 Top-k attention pruning

Based on the aforementioned hierarchical positional encoding, it becomes feasible to incorporate long-context inputs into LLMs. However, the incorporation of long-context inputs gives rise to a computational explosion in attention mechanisms, thereby presenting a non-trivial efficiency challenge. To this end, we propose the strategy of top-k attention pruning to eliminate redundant computation. The key idea is to guide the model's attention toward the top-k relevant documents while masking the remaining ones under the premise of attending to the beginning query. Figure 2 provides a brief explanation of top-k attention pruning.

Inspiration from top-k ranking. We take inspiration from the hierarchical generative process of top-k ranking in learning-to-rank [18]: (i) All top-k objects are selected and ranked before the other $n - k$ objects with query-object interactions. (ii) The full ordering on all the top-k objects is generated with listwise learning-to-rank algorithms [45]. When applying this process to listwise document reranking, we are motivated to retain both coarse-grained query-document interaction and fine-grained inter-document interaction, while simultaneously excluding redundant interaction during attention computation to improve efficiency.

Formulation of top-k attention pruning. Inspired by the hierarchical generative process, we formulate the process of top-k attention pruning as follows: (i) The leading k documents returned by the retriever, i.e., the first k documents in the listwise input, are identified as top-k relevant documents demanding fine-grained interaction. (ii) Tokens within each document are guaranteed to attend to the user query q at each time step. (iii) Inter-document attention interactions are retrained within top-k documents while eliminated within non-top-k documents. The above procedures can be succinctly summarized as follows: each token w_i^j within the document d_i can only attend to tokens within $q \oplus d_1 \oplus d_2 \oplus \cdots \oplus d_k \oplus d_i$ (before the current token), where $\oplus$ is the concatenation operation.

Implementation with attention masks. The top-k attention pruning mechanism can be efficiently implemented using chunk-aware attention masks. As mentioned previously, the listwise input comprises three categories of chunks: instruction chunks, query chunks, and document chunks. We set each chunk to a fixed length via padding and truncating, and chunks are delimited by a special token [SEP] to impose a structured input format. The listwise

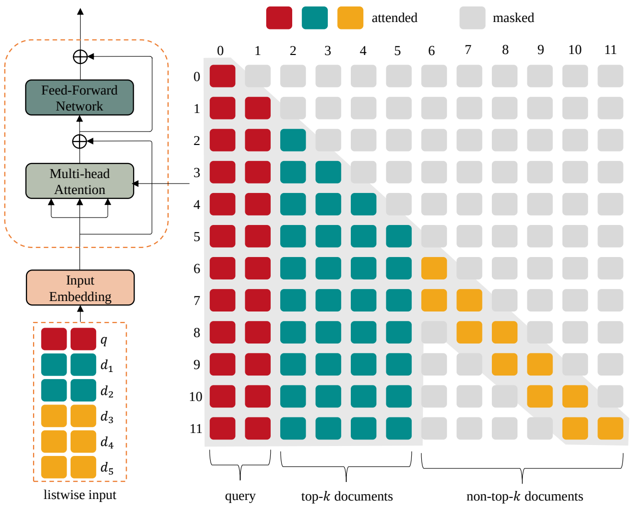


Figure 2: Top-k attention pruning. We assume that the length of both the query and each document is 2 for simplicity. We input 5 documents and perform top-2 attention pruning.

input can be decomposed into three spans. (i) The starting span allows each token to attend to the beginning query, thereby enabling coarse-grained query-document interaction across all documents. (ii) The middle span allows each token to attend to the top-k documents, hence allowing for fine-grained interaction within top-k relevant documents. (iii) The ending one allows each token to attend to recent tokens within the same document. The remaining tokens are masked to reduce computation costs. The attention mask can be seamlessly integrated into the attention function of each Transformer layer [42] during both fine-tuning and inference. In our implementation, we treat the instruction prompt as a special form of user query to ensure that the model accurately comprehends the instructions at each computational step.

Discussion. Han et al. [8] proposed a Λ-shaped attention mask to achieve zero-shot length generalization for LLMs. However, their approach diverges from ours in two primary respects: (i) They focus on natural language processing (NLP) tasks by modeling the attention mask at the token level, while we focus on listwise document reranking modeling the attention mask at the document level. (ii) In contrast to their implementation, which directly applies the attention mask only during inference, our approach integrates the attention mask throughout both the fine-tuning and inference phases. Our proposed top-k attention pruning distinguishes itself from other techniques by concentrating on the intrinsic attributes of document reranking.

3.3 Training and inference

To obtain high-quality listwise training labels, we utilize commercial LLMs such as DeepSeek [15] as the teacher models to produce the full rankings for given queries and related documents. Following [16], we employ a multi-pass sliding window approach to construct training data with listwise labels. The multi-pass sliding window approach iteratively applies the sliding window strategy to rerank the remaining $A - k(W - S)$ documents, with the round index k incremented from 0 until a complete ranking of all documents is obtained. Here, A represents the total number of input documents, W denotes the window size, and S indicates the step size.

Based on the constructed training data, we leverage the standard language modeling loss to optimize LongRanker:

$$\mathcal{L} = -\sum_{i=1}^{|O|} \log(P_\theta(O_i|S, O_{<i})), \tag{10}$$

where $S = [p, q, d_1, d_2, \ldots, d_i, \ldots, d_n]$ denotes the input prompt sequence and O denotes the output permutation. During inference, we directly generate a permutation of documents for the given query and documents in a single pass.

4 Experimental Setup

Datasets and evaluation metrics. We assess distinct models on passage reranking datasets of TREC 2019 and 2020 Deep Learning Tracks [4, 5], denoted as TREC-DL-19 and TREC-DL-20 for convenience. Both datasets are rooted in the MS MARCO v1 passage corpus, which consists of 8.8 million passages. We also employ the the BEIR benchmark, which comprises 18 datasets across different domains to assess the zero-shot capability of reranking models. Following [16, 36], we utilize 8 datasets from the BEIR benchmark for evaluation. We rerank the top-100 passages retrieved by the first-stage retriever BM25 [31] for all methods. We employ nDCG@10 as the evaluation metric to assess the reranking effectiveness. Following [55], we use two metrics to assess the efficiency of distinct models: (i) Latency: The average response latency for each query, measured in seconds. (ii) Prompt token number: The average number of tokens in the input prompt to the language models per query.

Implementation details. LongRanker is fine-tuned using training data distilled from DeepSeek-V3.2-Exp[1]. Specifically, we randomly sample 10k queries from the MS MARCO training set [17] and employ DeepSeek-V3.2-Exp to generate teacher labels for knowledge distillation. In the stage of training data construction, we utilize the sliding window strategy with the window size of 20 and the step size of 10. The backbone model architecture of LongRanker is Qwen2.5-1.5B-Instruct[2], which supports a context window of up to 128K tokens. We insert hierarchical positional encoding following the implementation of BiPE_RoPE [9]. The input prompt to LongRanker follows the format illustrated in Section 2.1. In the listwise input, each chunk (including instruction, query, and document chunks) is truncated or padded to the fixed length 150, and consecutive chunks are delimited by the special token `[SEP]`. LongRanker is finetuned for 3 epochs with a learning rate of 1e-5 and batch size of 1. All models are trained on 8 NVIDIA A100 GPUs and evaluated on 1 NVIDIA A100 GPU. For model inference, we set k in the top-k attention pruning component to 20. Each query only needs a single round of inference since the sliding window strategy is eliminated. All methods rerank top-100 documents retrieved by BM25. We employ Pyserini [14] to implement BM25. To assess the efficiency of LLM-based rerankers, we reimplement them in our working environment following the prompts recommended by the original literature.

Baselines. Detailed descriptions of the baseline models are provided in Appendix A.

[1]https://chat.deepseek.com
[2]https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct

5 Experimental Results

5.1 Main results

Table 1 compares different models in terms of effectiveness on TREC-DL-19, TREC-DL-20 and BEIR.

Comparison to unsupervised models. In terms of document reranking effectiveness, the nDCG@10 scores presented in Table 1 lead to the following findings: (i) LongRanker exhibits significant effectiveness gains against pointwise methods. The underlying reason might be that LongRanker can capture global inter-document interactions for all retrieved documents, whereas the pointwise method only captures query-document interactions. (ii) LongRanker achieves effectiveness on par with pairwise and setwise methods while containing substantially fewer parameters. The underlying reason might be that LongRanker contains the process of supervised fine-tuning while their methods are evaluated under zero-shot settings. (iii) Despite surpassing RankGPT$_{3.5}$ that is based on ChatGPT, LongRanker is inferior to RankGPT$_4$ across most benchmark datasets. We attribute this phenomenon to the fact that RankGPT$_4$ is a proprietary LLM comprising massive parameters, which empower it with powerful semantic understanding and language modeling capabilities. However, the high economic expenses of LLM API calls restricts the realistic utilization of RankGPT$_{3.5}$ and RankGPT$_4$.

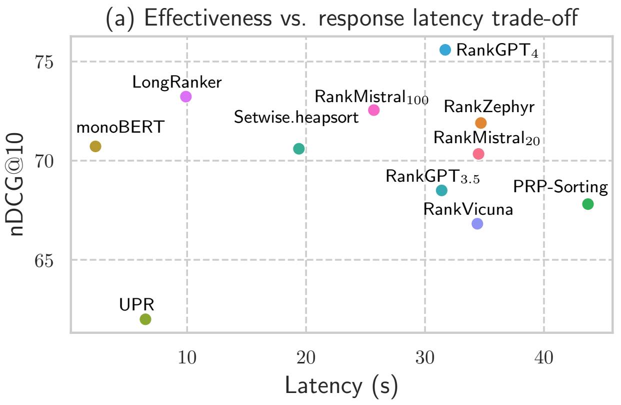
Comparison to supervised models. When it comes to the supervised models, the nDCG@10 scores presented in Table 1 lead to the following findings: (i) Compared to pointwise methods monoBERT and monoT5, LongRanker can achieve better reranking performance by a large margin on all benchmarks although all models utilize labeled training data. We attribute the sucess of LongRanker to the capture of global inter-document context. (ii) LongRanker, RankVicuna, RankZephyr and RankMistral$_{20}$ are all supervised listwise methods fine-tuned on listwise labels generated by proprietary LLMs. We can observe that LongRanker outperforms the remaining three models on most benchmark datasets with only 1.5B parameters. We attribute the effectiveness advantage to the elimination of sliding window strategy. LongRanker can capture global inter-document interactions by accommodating all 100 documents in one pass, whereas the remaining models are limited to interactions within a sliding window of size 20. (iii) Compared to RankMistral$_{100}$ that enables one-pass reranking as well, LongRanker still show effectiveness advantage on TREC-DL-19, TREC-DL-20 and 6 out of 8 BEIR datasets. The underlying reason might be that both the hierarchical positional encoding and top-k attention pruning in LongRanker are tailored for document reranking while RankMistral$_{100}$ is built on the Mistral backbone that has not been adapted to reranking at the model architecture level.

5.2 Effectiveness-efficiency trade-off

In this section, we assess the effectiveness-efficiency trade-off of the baseline models. As mentioned in Section 4, efficiency is evaluated using two metrics: response latency and prompt token number. For clarity of presentation, we report the relative prompt token number normalized with respect to LongRanker. For the pairwise PRP reranker, we only showcase the most efficient aggregation strategy, i.e., PRP-Sorting. Similarly, we only showcase the most efficient aggregation strategy for the setwise methods, i.e., Setwise.heapsort.

Table 1: Performance comparison on the benchmark datasets in terms of nDCG@10. All methods rerank top-100 documents retrieved by BM25. Sliding represents listwise methods with the sliding window strategy. One-pass represents listwise methods that reranks all candidate documents in a single pass. The best results within each part are highlighted in bold.

Models	Strategy	DL19	DL20	Covid	DBPedia	SciFact	NFCorpus	Signal	Robust04	Touche	News	Avg.
BM25	-	50.58	47.96	59.47	31.80	67.89	33.75	33.04	40.70	44.22	39.52	43.80
Unsupervised												
UPR (11B)	Pointwise	53.85	56.02	68.11	30.91	72.69	43.11	31.91	42.43	19.69	35.04	42.99
PRP-Allpair (11B)	Pairwise	69.87	69.85	-	-	-	-	-	-	-	-	-
PRP-Sorting (11B)	Pairwise	67.81	67.77	-	-	-	-	-	-	-	-	-
PRP-Sliding-10 (11B)	Pairwise	67.00	67.35	-	-	-	-	-	-	-	-	-
Setwise.heapsort (11B)	Setwise	70.60	68.80	75.20	40.2	72.60	34.60	32.10	51.30	29.70	47.30	47.90
Setwise.bubblesort (11B)	Setwise	71.10	68.60	76.80	42.40	**75.40**	34.60	34.30	53.40	**38.80**	47.90	50.50
RankGPT$_{3.5}$ (API)	Sliding	65.80	62.91	76.67	44.47	70.43	48.85	32.12	50.62	36.18	48.85	49.37
RankGPT$_4$ (API)	Sliding	**75.59**	**70.56**	**85.51**	**47.12**	74.95	**52.89**	**34.40**	**57.55**	38.57	**52.89**	**53.68**
Supervised												
monoBERT (340M)	Pointwise	70.72	67.28	73.45	41.69	62.22	34.92	30.63	44.21	30.26	47.03	45.55
monoT5 (220M)	Pointwise	70.58	67.33	75.94	42.43	65.07	35.42	31.20	44.15	30.35	46.98	46.44
RankVicuna (7B)	Sliding	67.72	65.98	79.19	**44.51**	70.67	34.51	**34.24**	48.33	33.00	47.15	48.95
RankZephyr (7B)	Sliding	73.39	70.02	**82.92**	44.42	75.42	38.26	31.41	53.73	30.22	**52.80**	51.15
RankMistral$_{20}$ (7B)	Sliding	70.34	69.58	80.86	42.52	75.82	38.38	33.90	54.69	**37.18**	51.42	51.85
RankMistral$_{100}$ (7B)	One-pass	72.55	71.29	82.24	43.54	77.04	39.14	33.99	57.91	34.63	50.59	52.40
LongRanker (1.5B)	One-pass	**73.63**	**72.18**	81.94	44.09	**77.26**	**39.28**	33.03	**58.09**	35.46	51.08	**52.53**



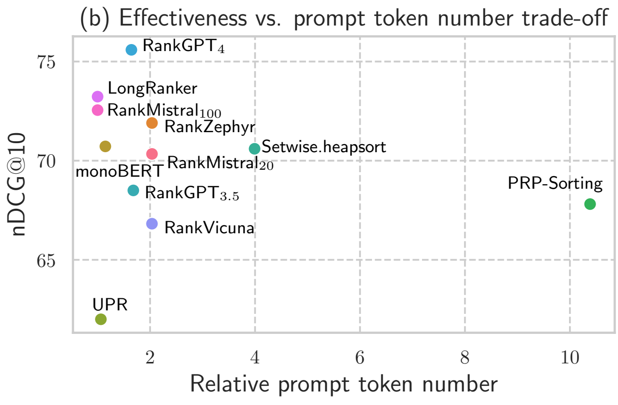


Figure 3: Effectiveness vs. response latency trade-off and effectiveness vs. prompt token number trade-off. Up and left is better. The relative prompt token number is with respect to LongRanker.

Moreover, we omit monoT5 since it exhibits similar effectiveness and efficiency distribution to monoBERT. Since all models exhibit consistent effectiveness and efficiency on the TREC-DL-19, TREC-DL-20 and BEIR datasets, we only report the effectiveness-efficiency trade-off on the TREC-DL-19 dataset, as shown in Figure 3.

Effectiveness vs. response latency. According to Figure 3(a), we can find that: (i) Despite the efficiency, pointwise methods including UPR and monoBERT demonstrate moderated effectiveness, hindering their potential in realistic search scenarios. (ii) The pairwise reranker PRP-sorting exhibits relatively poor effectiveness-efficiency balance with moderated nDCG@10 score and the longest response latency. The setwise method Setwise.heasort mitigates the efficiency problem of pariwise rerankers, while still maintaining longer response latency and weaker reranking effectiveness compared to LongRanker. (iii) All listwise methods using the sliding window strategy, including RankGPT$_{3.5}$, RankGPT$_4$, RankZephyr, RankVicuna and RankMistral$_{20}$, are trapped in long time latency, which highlights the efficiency advantage of LongRanker. Despite the high effectiveness, the response latency of RankGPT$_4$ is high due to the sliding window strategy, limiting its realistic applications. (iv) The one-pass methods, namely LongRanker and RankMistral$_{100}$, exhibit shorter response latency while maintaining considerable

Table 2: Ablation study on TREC-DL-19 and TREC-DL-20.

Model	TREC-DL-19 nDCG@10	TREC-DL-20 nDCG@10
LongRanker	**73.63**	**72.18**
w/o HPE	69.48	67.40
w/o TAP	67.97	66.35
w/o Both	63.60	61.80

reranking effectiveness. LongRanker achieves an even better balance between efficiency and performance due to its fewer parameters and the attention pruning component.

Effectiveness vs. prompt token number. As depicted in Figure 3(b), we can find that: (i) Similarly, LongRanker comprehensively achieves best balance between effectiveness and the prompt token numbers. (ii) Despite the effectiveness, RankGPT_4 demands far more input prompt tokens since each document is fed into LLMs twice except those within the beginning and the ending sliding window. This will incorporate expensive economic costs since GPT-4 is charged \$0.0300 per 1k tokens. (iii) Compared to listwise rerankers with sliding windows, LongRanker and RankMistral_{100} significantly decreases the prompt token numbers while maintaining considerable effectiveness due to the elimination of sliding windows.

5.3 Ablation study

Table 2 illustrates the results of the ablation study for LongRanker. All ablation variants are fine-tuned using the same training data as LongRanker. Notably, the w/o Both variant corresponds to the baseline of directly fine-tuning the Qwen2.5-1.5B backbone model on the training data without any additional components.

Impact of hierarchical positional encoding. We ablate the hierarchical positional encoding module from LongRanker and denote this variant as w/o HPE. As shown in Table 2, the ranking performance of LongRanker degrades significantly without hierarchical positional encoding, which validates the effectiveness of our proposed method in enhancing length extrapolation for listwise document input.

Impact of top-k attention pruning. To validate the effectiveness of top-k attention pruning, we eliminate the attention mask and denote this full attention variant as w/o TAP. Surprisingly, the full attention model involving both query-document and all inter-document interactions substantially underperforms LongRanker. The underlying reason might be that top-k attention pruning is built on the nature of document reranking, providing positive guidance for LLMs to model relevance relationships.

Table 3: Effect on the effectiveness and latency by varying k.

Value of k	0	20	40	60	80	100
nDCG@10	69.84	73.63	62.83	63.05	65.27	67.97
Latency	9.0	9.9	10.9	12.1	13.5	14.9

Impact of k in top-k attention pruning. We further analyze the impact of the hyperparameter k on the reranking effectiveness and efficiency by varying the value of k. The results are reported in Table 3, note that the aforementioned full attention variant is the case when k is set to 100. We can find that: (i) The response latency generally increases with k. (ii) Surprisingly, LongRanker exhibits relatively strong ranking capability when k is set to 0, indicating the ability of LongRanker to capture query-document interactions. (iii) LongRanker demonstrates optimal effectiveness when k is set to 20, whereas the effectiveness suffers from a significant drop when k is set between 20 and 100(i.e., $k \in \{40, 60, 80\}$). The underlying reason might be that the intermediate values of k can introduce interference from noisy documents.

5.4 Analysis on ranking inconsistency

To verify the advantages of LongRanker over sliding window strategies in terms of ranking consistency, we compare LongRanker with methods based on multi-pass sliding windows on TREC-DL-19. Specifically, we compare LongRanker to $\text{RankGPT}_{3.5}$ that adopts the sliding window strategy by varying the window and step size. To quantify ranking inconsistency, we measure the number of inconsistent document pairs among the shared documents in adjacent sliding windows. Specifically, if the relative order of a pair of shared documents differs between two overlapping windows, we count this as an inconsistent pair. As depicted in Table 4, we can observe that: (i) As the window size increases, the number of inconsistent document pairs grows significantly with decreasing ranking effectiveness. (ii) LongRanker completely avoids inconsistent document pairs since the ranking process only needs one pass to LLMs.

Table 4: Comparison on ranking inconsistency. #IP denotes the average number of inconsistency pairs per query.

Model	Window	Step	#IP	nDCG@10
$\text{RankGPT}_{3.5}$	20	10	67.42	67.05
$\text{RankGPT}_{3.5}$	40	20	105.12	65.51
$\text{RankGPT}_{3.5}$	60	30	158.14	65.03
LongRanker	-	-	**0**	**73.63**

5.5 Comparison with long-context LLMs

As shown in Table 5, we compare LongRanker with long-context LLMs of varying sizes under both zero-shot and fine-tuning settings. We select Qwen2.5-1.5B[3] and Qwen2.5-7B[4], both of which support a context window of 128k tokens. All configurations exclude the sliding window strategy and perform one-pass reranking over the top-100 retrieved documents. In the zero-shot setting, off-the-shelf models are directly employed for reranking without any adaptation. In the fine-tuned setting, models are trained using the same data as LongRanker. According to Table 5, we can find that: (i) Both model sizes exhibit modest zero-shot performance, indicating that long-context LLMs require task-specific fine-tuning to effectively perform listwise reranking. (ii) Fine-tuning yields substantial improvements over zero-shot baselines for both model sizes. Notably, Qwen2.5-7B-finetune achieves the best nDCG@10 of 74.01 on TREC-DL-19, demonstrating the effectiveness of knowledge distillation from proprietary LLMs for learning reranking capabilities. (iii) LongRanker achieves competitive one-pass reranking performance compared to fine-tuned long-context LLMs while maintaining significantly lower response latency (9.9s vs. 14.9-25.7s), highlighting its practical potential for real-world deployment.

[3]https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
[4]https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

Table 5: Comparison with long-context LLMs.

Model	TREC-DL-19 nDCG@10	TREC-DL-20 nDCG@10	Latency
Qwen2.5-1.5B-zeroshot	50.58	47.96	16.1
Qwen2.5-1.5B-finetune	63.60	61.80	14.9
Qwen2.5-7B-zeroshot	58.32	52.93	25.0
Qwen2.5-7B-finetune	**74.01**	67.75	25.7
LongRanker	73.63	**72.18**	**9.9**

5.6 Impact of document number

To further evaluate the generalization capability of LongRanker across varying numbers of passages, we adjust the number of documents retrieved by the first-stage retriever, denoted as N, and analyze its impact on reranking effectiveness. As illustrated in Table 6, LongRanker trained on listwise data with 100 documents demonstrates robust reranking performance across different document counts. Notably, the method achieves superior effectiveness when the document count is increased to 200, compared to smaller configurations such as 20 or 50. This observation suggests that LongRanker is adept at leveraging long-context information and is particularly well-suited for scenarios where extensive contextual information is available.

Table 6: Impact of document count N on reranking effectiveness.

Value of N	20	50	100	200
nDCG@10	66.09	69.68	73.63	71.05

6 Related Work

LLMs for reranking. Recent research [23, 34, 37, 52] underscores the efficacy of LLMs as proficient rerankers. Four distinct categories of prompting strategies are proposed to activate the reranking capabilities within LLMs, specifically including pointwise, pairwise, setwise and listwise prompting methods. Pointwise methods [3, 13, 32, 53, 54] utilize LLMs to generate a relevance score for the given query and each candidate document, typically following the relevance generation paradigm [13, 53] that prompts models to determine the query-document relevance and the query generation paradigm [3, 32, 54] that instructs LLMs to generate a relevant query for each document. Pair-wise methods [24, 27] prompt LLMs to compare the relevance of a pair of documents to a given query. Various comparison strategies, such as all-pair comparisons, sorting-based, and sliding windows, are used to aggregate these pairwise preferences into a final ranked list. Setwise methods[55] select the most relevant document to the query from multiple documents in one pass, and employ comparison strategies to aggregate the comparison results as well. Listwise methods [21, 25, 29, 30, 33, 36, 38] input a query along with candidate documents into LLMs, which are then instructed to generate a ranked permutation based on relevance. However, the limited context window of LLMs restricts direct reranking of entire document lists, hence a sliding window strategy is usually adopted. Despite the effectiveness, sliding windows can lead to redundant comparisons, Liu et al. [16] detected the efficiency dilemma by conducting a comprehensive study on long-context LLMs for ranking. However, the proposed enhancement method neglects the inherent characteristics of listwise document reranking to some degree. Gupta et al. [7] identified the potential of exploiting the structure inherent in listwise document reranking and introduced BlockRank, an efficient in-context ranking method.

Long-context LLMs. Extensive efforts have been conducted to extend the context window of LLMs. When it comes to longer input than encountered during the pre-training phase, LLMs often struggle to generalize effectively to these extended inputs due to the OOD problem of relative distances. To address this issue, many studies [2, 9, 22, 46, 50] have been dedicated to adapting the positional encoding to improve robustness in generalizing to lengthy input sequences. Wherein He et al. [9] first proposed BiPE to combine absolute and relative positional encoding, thereby boosting length extrapolation capabilities. Another challenge LLM faces during length extrapolation is the excessive complexity in attention computation. To address this limitation, Beltagy et al. [1] proposed three categories of attention patterns to achieve a linear scaling with sequence length. Guo et al. [6] also introduces a novel transient global attention mechanism that allows each token to attend to its neighborhood as well as all global tokens. Similarly, Han et al. [8] designed a Λ-shaped attention mask along with a distance ceiling technique to improve the ability of LLMs to process long contexts. Despite the promising empirical results, directly applying these methods to the reranking task overlooks the intrinsic nature of document reranking. In this paper, we shift our focus toward exploring how to transform LLMs into efficient long-context rerankers.

7 Conclusion

In this work, we explore how to perform efficient one-pass document reranking with long-context LLMs. We devise the hierarchical positional encoding component to improve length extrapolation for listwise inputs, and propose the top-k attention pruning approach to lower the computation costs. Our main findings are that our proposed LongRanker method achieves significant advantages in terms of effectiveness and efficiency when compared with other LLM-based rerankers.

An important limitation of our work is that the training data construction for LongRanker depends on proprietary LLMs, which incurs additional training costs. We plan to explore proprietary LLM-free training pipelines in future work. Another limitation concerns the evaluation metrics, which focus solely on the reranking performance of the top-10 documents in the ranked list. This approach is insufficient to comprehensively assess the full reranking capability of long-context LLMs across the entire document set.

Acknowledgments

This work was funded by the Strategic Priority Research Program of the CAS under Grant No. XDB0680102, the National Natural Science Foundation of China under Grants No. 62472408, U25B2076, 62372431 and 62441229, the National Key Research and Development Program of China under Grants No. 2023YFA1011602. This research was also (partially) supported by the Dutch Research Council (NWO), under project numbers 024.004.022, NWA.1389.20.183, and KICH3.LTP.20.006, and the European Union under grant agreements No. 101070212 (FINDHR) and No. 101201510 (UNITE). All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[2] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595* (2023).

[3] Sukmin Cho, Soyeong Jeong, Jeong yeon Seo, and Jong C Park. 2023. Discrete Prompt Optimization via Constrained Generation for Zero-shot Re-ranker. In *Findings of the Association for Computational Linguistics: ACL 2023*. 960–971.

[4] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. arXiv:2102.07662 [cs.IR] https://arxiv.org/abs/2102.07662

[5] 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).

[6] Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. LongT5: Efficient text-to-text transformer for long sequences. *arXiv preprint arXiv:2112.07916* (2021).

[7] Nilesh Gupta, Chong You, Srinadh Bhojanapalli, Sanjiv Kumar, Inderjit Dhillon, and Felix Yu. 2025. Scalable In-context Ranking with Generative Models. *arXiv preprint arXiv:2510.05396* (2025).

[8] Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137* (2023).

[9] Zhenyu He, Guhao Feng, Shengjie Luo, Kai Yang, Di He, Jingjing Xu, Zhi Zhang, Hongxia Yang, and Liwei Wang. 2024. Two Stones Hit One Bird: Bilevel Positional Encoding for Better Length Extrapolation. *arXiv preprint arXiv:2401.16421* (2024).

[10] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[11] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[12] Mei Kobayashi and Koichi Takeda. 2000. Information retrieval on the web. *ACM computing surveys (CSUR)* 32, 2 (2000), 144–173.

[13] 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).

[14] 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*. 2356–2362.

[15] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[16] Wenhan Liu, Xinyu Ma, Yutao Zhu, Ziliang Zhao, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. 2024. Sliding Windows Are Not the End: Exploring Full Ranking with Long-Context Large Language Models. *arXiv preprint arXiv:2412.14574* (2024).

[17] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. [n. d.]. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *choice* 2640 ([n. d.]), 660.

[18] Shuzi Niu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2012. A new probabilistic model for top-k ranking problem. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2519–2522.

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

[20] 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.

[21] Andrew Parry, Sean MacAvaney, and Debasis Ganguly. 2024. Top-down partitioning for efficient list-wise ranking. *arXiv preprint arXiv:2405.14589* (2024).

[22] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071* (2023).

[23] Ronak Pradeep, Yuqi Liu, Xinyu Zhang, Yilin Li, Andrew Yates, and Jimmy Lin. 2022. Squeezing water from a stone: a bag of tricks for further improving cross-encoder effectiveness for reranking. In *European Conference on Information Retrieval*. Springer, 655–670.

[24] 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).

[25] 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).

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

[27] 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).

[28] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. 2024. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*. 1504–1518.

[29] Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. 2024. FIRST: Faster Improved Listwise Reranking with Single Token Decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 8642–8652.

[30] Ruiyang Ren, Yuhao Wang, Kun Zhou, Wayne Xin Zhao, Wenjie Wang, Jing Liu, Ji-Rong Wen, and Tat-Seng Chua. 2025. Self-Calibrated Listwise Reranking with Large Language Models. In *Proceedings of the ACM on Web Conference 2025*. 3692–3701.

[31] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[32] 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*. 3781–3797.

[33] Ferdinand Schlatt, Maik Fröbe, Harrisen Scells, Shengyao Zhuang, Bevan Koopman, Guido Zuccon, Benno Stein, Martin Potthast, and Matthias Hagen. 2025. Set-encoder: Permutation-invariant inter-passage attention for listwise passage re-ranking with cross-encoders. In *European Conference on Information Retrieval*. Springer, 1–19.

[34] Clemencia Siro, Mohammad Aliannejadi, and Maarten de Rijke. 2023. Understanding and Predicting User Satisfaction with Conversational Recommender Systems. *ACM Transactions on Information Systems* 42, 2 (2023), 1–37.

[35] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing* 568 (2024), 127063.

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

[37] Raphael Tang, Crystina Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Türe. 2024. Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 2327–2340.

[38] 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. *arXiv preprint arXiv:2310.07712* (2023).

[39] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/

[40] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663* (2021).

[41] 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).

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[43] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Large search model: Redefining search stack in the era of llms. In *ACM SIGIR Forum*, Vol. 57. ACM New York, NY, USA, 1–16.

[44] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG type ranking measures. In *Conference on learning theory*. PMLR, 25–54.

[45] Fen Xia, Tie-Yan Liu, and Hang Li. 2009. Statistical consistency of top-k ranking. *Advances in Neural Information Processing Systems* 22 (2009).

[46] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. 2023. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039* (2023).

[47] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang,

Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671* (2024).

[48] Yifan Zeng, Ojas Tendolkar, Raymond Baartmans, Qingyun Wu, Lizhong Chen, and Huazheng Wang. 2024. LLM-RankFusion: Mitigating Intrinsic Inconsistency in LLM-based Ranking. *arXiv preprint arXiv:2406.00231* (2024).

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

[50] Yikai Zhang, Junlong Li, and Pengfei Liu. 2024. Extending LLMs' Context Window with 100 Samples. *arXiv preprint arXiv:2401.07004* (2024).

[51] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[52] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (2023).

[53] Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Berdersky. 2023. Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels. *arXiv preprint arXiv:2310.14122* (2023).

[54] 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*. 8807–8817.

[55] 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*. 38–47.

Appendix

A Details of baseline models

In this section, we briefly introduce the details of baseline models. We generally categorize the ranking models into unsupervised and supervised methods.

For unsupervised models, we compare our method with the following baselines across distinct strategies: (i) UPR [32] is a LLM-based pointwise method that judges query-document relevance based on query generation. The backbone model is FLAN-T5-XXL with 10B parameters. (ii) PRP [28] is a LLM-based pairwise method that judge inter-document relevance and then aggregate the comparison results. Following [28], we incorporate three categories of comparison strategies, i.e., PRP-Allpair, PRP-Sorting, PRP-Sliding-10. The backbone model is FLAN-T5-XXL with 10B parameters. (iii) Setwise rerankers [55] first judge the most related document from a document set and then aggregate the comparison results. Following [28], we incorporate two comparison strategies, i.e., Setwise.heapsort and Setwise.bubblesort. The backbone model is FLAN-T5-XXL with 10B parameters. (iv) Listwise rerankers prompt LLMs to generate permutations of document identifiers. We compare with RankGPT$_{3.5}$ and RankGPT$_4$ [36] that utilize the proprietary LLM `gpt-3.5-turbo` and `gpt-4` as the backbone model, respectively. Both models employ a sliding window strategy with window size 20 and step 10 to yield the final ranked list.

For supervised models, we compare our method with the following baselines across distinct strategies: (i) monoBERT [19] is a pointwise cross-encoder reranker that fine-tunes BERT with the MS MARCO dataset [17]. (ii) monoT5 [20] is a pointwise reranker that fine-tunes the sequence-to-sequence model T5 to produce the words "true" or "false" revealing the query-document relevance. (iii) RankVicuna [25] is a listwise model that uses ranked lists generated by RankGPT$_{3.5}$ to fine-tune Vicuna. (iv) RankZephyr [26] is a listwise model that distills from both RankGPT$_{3.5}$ and RankGPT$_4$ with Zephyr. Both RankVicuna and RankZephyr employ a sliding window strategy with window size 20 and step 10 to yield the final ranked list. (v) RankMistral [16] is a listwise model that fine-tunes Mistral-7B-Instruct-v0.3 [11] with rank lists distilled from `gpt-4o`. RankMistral$_{20}$ employs a sliding window strategy with window size 20 and step 10 to yield the final ranked list. RankMistral$_{100}$ enables one-pass ranking of all candidate documents.