Sliding Windows Are Not the End: Exploring Full Ranking with Long-Context Large Language Models

Wenhan Liu[1], **Xinyu Ma**[2], **Yutao Zhu**[1], **Ziliang Zhao**[1], **Shuaiqiang Wang**[2]
Dawei Yin[2] and **Zhicheng Dou**[1,3*]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]Baidu Inc., Beijing, China
[3]Beijing Key Laboratory of Research on Large Models and Intelligent Governance
lwh@ruc.edu.cn, xinyuma2016@gmail.com, dou@ruc.edu.cn

Abstract

Large Language Models (LLMs) have shown exciting performance in listwise passage ranking. Due to the limited input length, existing methods often adopt the *sliding window* strategy. Such a strategy, though effective, is inefficient as it involves repetitive and serialized processing, which usually re-evaluates relevant passages multiple times. As a result, it incurs redundant API costs, which are proportional to the number of inference tokens. The development of long-context LLMs enables the *full ranking* of all passages within a single inference, avoiding redundant API costs. In this paper, we conduct a comprehensive study of long-context LLMs for ranking tasks in terms of efficiency and effectiveness. Surprisingly, our experiments reveal that full ranking with long-context LLMs can deliver superior performance in the supervised fine-tuning setting with a huge efficiency improvement. Furthermore, we identify two limitations of fine-tuning the full ranking model based on existing methods: (1) the sliding window strategy fails to produce a full ranking list as a training label, and (2) the language modeling loss cannot emphasize top-ranked passage IDs in the label. To alleviate these issues, we propose a complete listwise label construction approach and a novel importance-aware learning objective for full ranking. Experiments show the superior performance of our method over baselines. Our codes are available at `https://github.com/RUC-NLPIR/fullrank`.

1 Introduction

In recent years, large language models (LLMs) have demonstrated impressive zero-shot capabilities in passage ranking tasks (Sun et al., 2023; Zhu et al., 2023). The listwise ranking approach, which processes multiple passages simultaneously and directly outputs a reranked list of passage IDs, has been widely adopted and shown to outperform other methods. For instance, Sun et al. (2023) achieved state-of-the-art performance using the proprietary GPT-4 model on both the TREC (Craswell et al., 2020b) and BEIR (Thakur et al., 2021) benchmarks. Furthermore, several studies (Pradeep et al., 2023a,b) also attempt to distill the listwise ranking capabilities of proprietary models into more moderately sized LLMs to enhance ranking efficiency and improve the reproducibility of results.

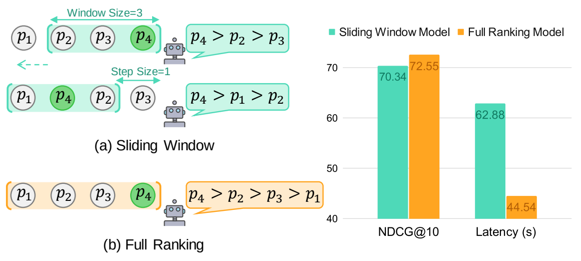


Figure 1: The sliding window strategy and full ranking strategy are shown in part (a) and part (b), respectively. The bar chart shows the comparison between our fine-tuned sliding window model and full ranking model in terms of NDCG@10 and latency (per query) on the TREC DL19 dataset.

Due to the limited context length of many LLMs, existing listwise ranking methods can only process a subset of passages at a time, ultimately relying on a sliding window strategy (Sun et al., 2023) to rank the entire passage list (see part (a) in Figure 1). Based on fixed window size and step size, this strategy promotes relevant passages that are initially lowly ranked to the top. However, in this strategy, many passages are evaluated multiple times due to the overlapping parts of adjacent windows, leading to significant redundancy and increased API costs, which grow proportionally with the number of inference tokens. Additionally, the sequential dependency between windows also limits inference parallelization, resulting in efficiency bottlenecks.

[*]Corresponding author.

Recently, with the development of long-context techniques (Chen et al., 2023; Ding et al., 2023), some LLMs, such as Mistral-7B-Instruct-v0.3 (32k) and LLaMA 3.1-8B-Instruct (128k), have supported longer input lengths. This enables the input of all retrieved passages and the output of the full ranked list in a single step, which we refer to as *full ranking* in this paper (see part (b) in Figure 1). Full ranking not only eliminates the repetitive and time-consuming sliding window process but also reduces redundant passage inference, thereby significantly lowering API costs. Even though, the effectiveness and efficiency of the full ranking strategy remain unclear. As for effectiveness, although full ranking allows for more global interactions among passages, which may lead to higher ranking accuracy, the increased input length also brings higher difficulty for LLMs. As for efficiency, while the full ranking eliminates the sliding window's serial processing, the dramatically increased input length also increases the time cost of LLM encoding. We believe that a thorough investigation of these questions will facilitate the application of long-context LLMs to ranking tasks.

In this paper, we conduct a comprehensive study of long-context LLMs for ranking. Specifically, we compare the full ranking and sliding window strategies in terms of efficiency and effectiveness, and draw the following conclusions: (1) In the zero-shot setting, the full ranking model demonstrates higher efficiency but lower effectiveness compared to the sliding window model. (2) In the supervised fine-tuning setting, the full ranking model outperforms the sliding window model, demonstrating the advantage of full ranking with proper fine-tuning.

Furthermore, we identify the limitations of applying existing listwise training methods (Pradeep et al., 2023a,b) to fine-tune a full ranking model: (1) Existing methods for ranking a passage list are primarily based on sliding windows, passing through the list from the back to the front. Theoretically, this approach can only guarantee the order of the top-ranked passages, rather than generating a full ranking list. (2) In terms of optimization objectives, existing methods use a standard language modeling loss that applies an equal penalty to all passage IDs in the training label. However, the training label for a full ranking model contains hundreds of passage IDs, with only a few of the top-ranked passage IDs being relevant. This leads to a significant imbalance between relevant and irrelevant passages. As a result, the penalty on relevant passage IDs is overshadowed by that on irrelevant ones, which conflicts with the evaluation of the ranking task that prioritizes top-ranked passages.

To address these limitations, we first design a multi-pass sliding window approach, which iteratively generates a full ranking list as a training label, overcoming the limitation of producing only the top-ranked passages in a single pass. Then, we propose an importance-aware loss that reweights the passage IDs in the label based on their rank position, ensuring that higher-ranked passage IDs receive more attention. Extensive experiments on TREC and BEIR benchmarks demonstrate that our fine-tuned full ranking model outperforms previous state-of-the-art models, as well as our fine-tuned sliding window model. Figure 1 shows the advantage of our fine-tuned full ranking model compared to the sliding window model: an absolute improvement of 2.2 in terms of NDCG@10 and a reduction in latency by 29.3% on the TREC DL19 dataset.

Our contributions are summarized as follows:

• We conduct a comprehensive study of long-context LLMs for ranking. To the best of our knowledge, we are the first to investigate the application of long-context LLMs in ranking tasks.

• We reveal that, in a zero-shot setting, full ranking with long-context models is more efficient but less effective compared to the sliding window strategy.

• To enhance the effectiveness of full ranking, we propose a complete listwise label construction approach and a novel importance-aware learning objective for fine-tuning the full ranking model.

2 Listwise Ranking with LLMs

In this section, we discuss the fundamentals of listwise ranking in zero-shot and supervised fine-tuning settings, respectively.

2.1 Zero-shot Ranking

Given a user query q and a list of passages $P = [p_1, \ldots, p_N]$, the listwise ranking task takes q and P as the input, and outputs a ranked sequence of IDs indicating the relevance of each passage (*i.e.*, [3] > [1] > ...). The prompt we use is shown in Appendix A. In this paper, we use two different ranking strategies based on long-context LLMs: (1) the full ranking strategy and (2) the sliding window strategy. The full ranking involves inputting the entire passage list P into the LLM, enabling it to rank all passages simultaneously in a single step. The sliding window strategy uses a window of size

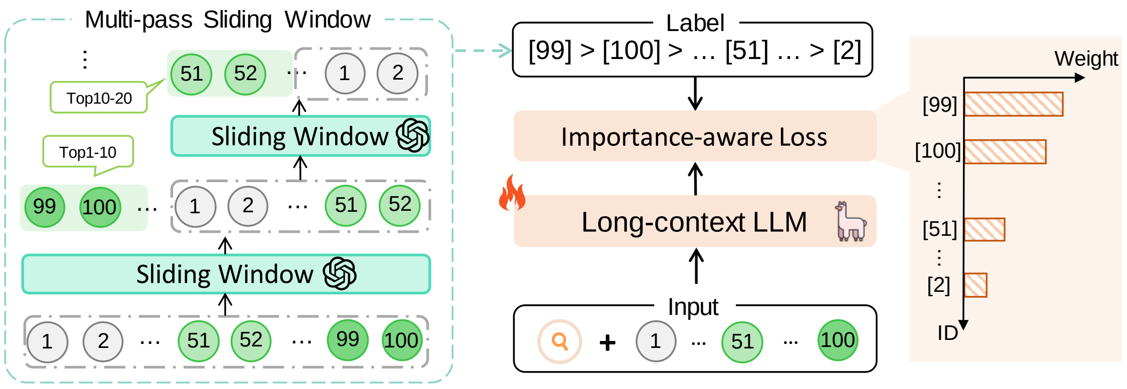


Figure 2: The training method of the full ranking model. We first use a multi-pass sliding window approach to iteratively obtain the full ranking list of passages. Then, we design an importance-aware loss that assigns different weights to the IDs in the label for model optimization.

w, sliding from the end of the passage list to the beginning with a step size s. In this paper, following previous studies (Sun et al., 2023; Pradeep et al., 2023a), we set passage number N, window size w, and step size s to 100, 20, and 10, respectively.

2.2 Supervised Fine-tuning

Existing studies (Pradeep et al., 2023a,b) on fine-tuning listwise rerankers are primarily based on distillation techniques: the teacher model (*e.g.*, GPT-4) receives a list of passages and outputs a sequence of passage IDs, which will be used as the supervised label for fine-tuning. The training process then optimizes the listwise reranker by minimizing the standard language modeling loss $\mathcal{L}$:

$$\mathcal{L} = -\sum_{i=1}^{|y|} \log(P_\theta(y_i \mid x, y_{<i})), \quad (1)$$

where x represents the input prompt and y is the teacher label, which is primarily obtained by reranking the top-20 passages retrieved by BM25.

3 Fine-tuning Full Ranking Model

As we mentioned in Section 1, directly applying existing listwise training methods to train a full ranking model has two limitations: (1) One-pass sliding window process can only guarantee the top ranking, failing to obtain the full ranking list, similar to how a single pass of bubble sort algorithm can only guarantee the top-1 ranked item. Directly applying a full ranking to all passages is reasonable, but it is less effective than the sliding window strategy in a zero-shot setting, which will be discussed in Section 4.2. (2) Given a full ranking list, which exhibits a significant imbalance between relevant and irrelevant passage IDs, the standard language modeling loss applies equal penalties to each ID, causing the top-ranked IDs to be overwhelmed. To address these limitations, we design a multi-pass sliding window approach to generate high-quality full ranking lists as training labels and propose a novel importance-aware learning objective for model optimization. Next, we will introduce the details of our methods.

3.1 Multi-pass Sliding Window Approach

To generate the full ranking list for model fine-tuning, we first employ BM25 to retrieve the top-100 candidate passages, which are then reranked using a teacher model. In theory, a single pass of the sliding window process (ranking from back to front with a window size of 20 and a stride of 10) can only ensure the retrieval of the top 10 most relevant passages. To overcome this limitation, we propose a multi-pass sliding window approach to obtain the full ranking list, as illustrated in Figure 2. In the first pass, the sliding window strategy is applied to rerank all 100 passages, yielding the top 10 most relevant passages. In the second pass, the same strategy is used to rerank the remaining 90 passages, producing the next 10 most relevant passages. This iterative process continues until the entire reranked list is constructed.

3.2 Importance-Aware Learning Objective

The full ranking label contains up to 100 passage IDs, with a very small proportion of relevant passage IDs. Using the standard language modeling loss leads to the loss contributions of top-ranked relevant passage IDs being overshadowed by others. This misaligns with the evaluation of the ranking task, where more relevant passages are of greater importance and have a larger impact on ranking metrics. For example, "[99]" should be assigned the highest importance than other ids in the label "[99] > [100] > $\cdots$" while calculating the loss. To address this issue, we propose an importance-aware loss function $\mathcal{L}_{\text{ia}}$ which includes a position-based weight w_p to reweight the importance of each passage ID in the label. The $\mathcal{L}_{\text{ia}}$ is defined as:

$$\mathcal{L}_{\text{ia}} = -\sum_{i=1}^{|y|} w_i \log(P_\theta(y_i \mid x, y_{<i})), \tag{2}$$

$$w_i = \begin{cases} 1 + \frac{1}{\log_2(p_i+1)}, & i \in \text{passage IDs}, \\ \alpha, & i \notin \text{passage IDs}. \end{cases} \tag{3}$$

Here, p_i represents the passage rank corresponding to the i-th token. Note that each passage ID will be split into multiple tokens (*e.g.*, [99] will be split into "[", "9", "9", and "]"), and these tokens have the same weight.[1] Besides, we believe that the importance of passage IDs is higher than that of relational operator ">", so we set the weight of ">" as α ($\alpha <= 1$). By incorporating the importance-aware loss $\mathcal{L}_{\text{ia}}$, we ensure that higher-ranked passage IDs receive greater weight during loss calculation, which better aligns with the training of the full-ranking model.

4 Experiments

4.1 Setting

Evaluation Datasets For evaluation, we employed datasets from TREC DL 2019 (Craswell et al., 2020b) and TREC DL 2020 (Craswell et al., 2020a), as well as BEIR (Thakur et al., 2021) benchmark. BEIR includes 18 datasets from various domains, designed to assess the zero-shot ability of ranking models. Following previous studies (Sun et al., 2023), we choose 8 BEIR datasets for model evaluation. We rerank the top-100 passages retrieved by BM25 and use NDCG@10 as the evaluation metric.

Baselines In addition to comparing the performance of sliding window model and full ranking model, we also include several finetuned rerankers for comparison: monoBERT (340M) (Nogueira and Cho, 2019), monoT5 (220M) (Nogueira et al., 2020), RankT5 (3B) (Zhuang et al., 2023b), RankVicuna (Pradeep et al., 2023a) and RankZephyr (Pradeep et al., 2023b). MonoBERT, monoT5, and RankT5 are trained on annotated query-passage pairs from the MS MARCO training set, while RankVicuna and RankZephyr are distilled from ranking lists generated by proprietary models, namely ChatGPT and GPT-4. The detailed descriptions of the baselines are presented in Appendix E.

Implementation Details In our experiments, we evaluate the performance of the full ranking strategy based on long-context LLMs under two different settings: zero-shot and supervised fine-tuning, and compare with the sliding window strategy.

For the zero-shot setting, we select both open-source and proprietary models to provide a comprehensive evaluation. The open-source models include Mistral-7B-Instruct-v0.3 (32k), LLaMA3.1-8B-Instruct (128k), and Qwen2.5-7B-Instruct (128k). These models have been chosen for their ability to handle long contexts and strong performance on a wide range of tasks. We also compare models of different sizes, and the results can be found in the appendix F. We also include two proprietary models, GPT-4o-mini and GPT-4o (both have 128k context)[2].

For the supervised fine-tuning setting, we finetune a sliding window model for comparison with the full ranking model. Following previous studies (Pradeep et al., 2023a,b), we use BM25 to retrieve top-20 passages and rerank them using a teacher model to obtain the training label. We also experiment with sampling 20 passages from the top-100 retrieved ones as training passages, but find the performance is inferior to directly using the top-20 passages. Detailed experimental results are presented in Table 8. We choose Mistral-7B-Instruct-v0.3 as our backbone model and use two different teacher models: GPT-4o-mini and GPT-4o. The finetuned full ranking model and sliding window model are denoted as **RankMistral$_{\mathbf{100}}$** and **RankMistral$_{\mathbf{20}}$**, respectively. Both models are optimized with our proposed importance-aware loss for fair comparison. The detailed training configuration and API Cost for generating training labels can be found in appendix B and D, respectively.

[1]We also explore adding the passage IDs as new tokens into the LLM's vocabulary. Unfortunately, this strategy does not bring performance improvements.

[2]The version of GPT-4o-mini and GPT-4o we used is gpt-4o-mini-2024-07-18 and gpt-4o-2024-08-06, respectively.

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
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
RankT5 (3B)	Pointwise	72.29	69.14	79.03	44.85	74.60	38.19	31.73	50.52	34.66	49.11	50.34
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
RankZepyer (7B)	Sliding	73.39	70.02	82.92	44.42	75.42	38.26	31.41	53.73	30.22	52.80	51.15
Zero-shot												
Mistral-v0.3 (7B)	Sliding	62.55	58.48	74.79	40.66	56.61	34.68	30.09	45.61	34.41	44.40	45.16
	Full	45.33	47.86	62.63	33.85	51.58	32.71	27.63	37.38	37.54	37.76	40.14
Llama-3.1 (8B)	Sliding	65.18	59.49	74.68	39.23	56.41	35.33	28.07	45.86	36.78	44.98	45.17
	Full	53.50	52.17	60.90	34.78	61.01	33.20	31.37	39.42	**41.83**	41.61	43.02
Qwen-2.5 (7B)	Sliding	68.34	64.89	79.76	40.87	71.32	37.97	30.66	52.87	32.02	49.49	49.37
	Full	57.44	54.42	65.06	35.57	61.49	33.75	27.35	37.60	26.91	37.55	40.66
GPT-4o-mini	Sliding	72.36	67.30	80.03	44.54	73.14	38.73	33.64	57.41	30.91	50.91	51.16
	Full	68.80	63.02	77.70	41.97	71.42	37.35	32.35	52.32	31.37	47.33	48.98
GPT-4o	Sliding	**74.78**	69.52	**83.41**	**45.56**	**77.41**	**39.67**	**34.20**	**60.25**	32.26	**51.92**	**53.09**
	Full	73.94	**70.03**	82.10	43.31	74.85	39.00	32.63	55.95	30.42	47.96	50.78
SFT from GPT-4o-mini												
RankMistral$_{20}$	Sliding	69.08	66.31	80.37	43.46	72.08	37.46	32.97	54.70	33.74	48.84	50.45
RankMistral$_{100}$	Full	**73.17**	**70.16**	**82.57**	**44.54**	**75.47**	**38.73**	**33.49**	**56.36**	**38.77**	**51.08**	**52.63**
SFT from GPT-4o												
RankMistral$_{20}$	Sliding	70.34	69.58	80.86	42.52	75.82	38.38	33.90	54.69	**37.18**	**51.42**	51.85
RankMistral$_{100}$	Full	**72.55**	**71.29**	**82.24**	**43.54**	**77.04**	**39.14**	**33.99**	**57.91**	34.63	50.59	**52.40**
w/o L_{ia} (SFT from GPT-4o)												
RankMistral$_{20}$	Sliding	71.14	68.49	80.80	42.53	**76.15**	38.54	33.12	55.63	31.62	**50.86**	51.16
RankMistral$_{100}$	Full	**73.19**	**71.19**	**81.24**	**42.71**	75.82	**38.85**	**34.0**	**57.40**	**33.21**	50.27	**51.69**

Table 1: Results (NDCG@10) on TREC and BEIR. The best results among each part are marked in bold respectively. Avg. represents the averaged result of the 8 BEIR datasets.

4.2 Effectiveness Analysis

In this section, we evaluate the performance of the full ranking strategy and the sliding window strategy based on several long-context LLMs in zero-shot and fine-tuning settings. The results are shown in Table 1. For the three open-source long-context LLMs, we use abbreviations: Mistral-v0.3 (Mistral-7B-Instruct-v0.3), LLaMA-3.1 (LLaMA-3.1-8B-Instruct), and Qwen-2.5 (Qwen2.5-7B-Instruct). "Sliding" and "Full" refer to the sliding window and full ranking strategies, respectively. Next, we will provide a detailed analysis of both settings.

Zero-shot In the zero-shot setting, the full ranking strategy underperforms the sliding window strategy on nearly all datasets across all long-context LLMs. This demonstrates that full ranking significantly increases the ranking difficulty of LLM, resulting in a performance drop. However, there are some cases where the full ranking strategy outperforms the sliding window strategy. For instance, on the Touche dataset, the full ranking strategy yields better results for Mistral-v0.3, LLaMA-3.1, and GPT-4o-mini. This may be due to the dataset characteristics, where global interactions among passages play a larger role in assessing the relevance.

Besides, among the evaluated models, GPT-4o stands out by achieving the best full ranking performance. Notably, it achieves an average NDCG@10 of 71.99 on TREC and 50.78 on BEIR, proving its strong capability for processing long-context input in ranking tasks.

Supervised Fine-tuning In supervised fine-tuning setting, we find that our full ranking model RankMistral$_{100}$ outperforms sliding window model RankMistral$_{20}$ on nearly all datasets. Notably, RankMistral$_{100}$ achieves an average improvement of about 4 and 2 points on TREC and BEIR, respectively, compared to RankMistral$_{20}$ when fine-tuned from GPT-4o-mini. These results indicate that task-specific fine-tuning can help the full ranking model better model the global interaction between all the passages, thus yielding better ranking performance than the sliding window model. Additionally, we also experiment with using Qwen2.5-7B-Instruct as the backbone model and observe the same conclusion. Detailed results

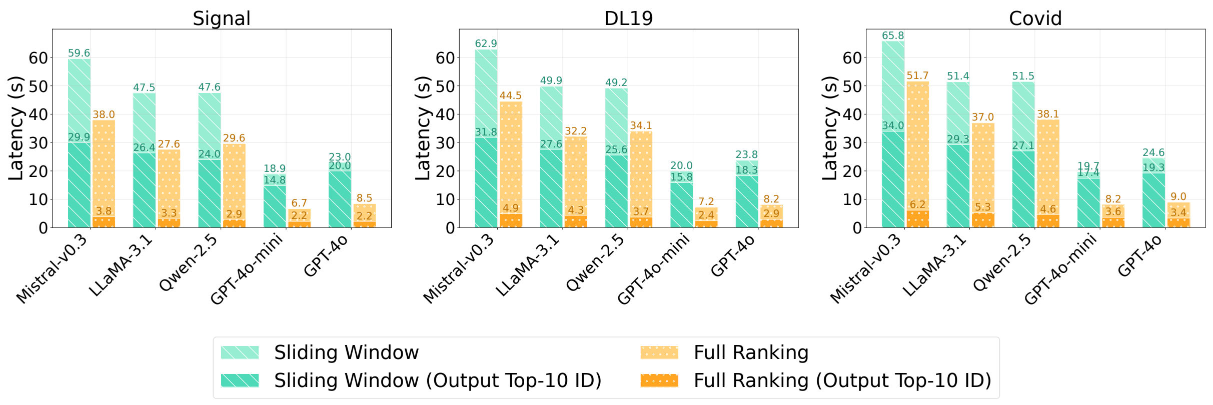


Figure 3: Latency of ranking top-100 passages based on full ranking and sliding window strategy. "Output Top-10 ID" indicates that the LLM only generates the top-10 ranked passage IDs.

are shown in Table 9.

Furthermore, we conduct an ablation study on RankMistral$_{20}$ and RankMistral$_{100}$ to prove the effectiveness of our importance-aware loss L_{ia}. Both models are fine-tuned from GPT-4o and trained with standard language modeling loss. The results are shown in part "w/o L_{ia} (SFT from GPT-4o)". After removing L_{ia}, both RankMistral$_{20}$ and RankMistral$_{100}$ drop about 0.7 points on BEIR Avg, proving that L_{ia} makes the model focus more on top-ranked passage IDs, thereby enhancing its ranking effectiveness. Besides, even trained with standard language modeling loss, RankMistral$_{100}$ still outperforms RankMistral$_{20}$ on dl19, dl20, and BEIR Avg, which further demonstrates the effectiveness advantage of RankMistral$_{100}$ in supervised fine-tuning setting.

Lastly, we also investigate the performance of RankMistral$_{100}$ under different ranking settings (*i.e.*, different initial passage order and ranking numbers). Due to limited space, we present the detailed analysis in Appendix C.

4.3 Efficiency Analysis

As discussed in Section 1, the full ranking strategy eliminates the redundant and time-consuming sliding window operations. However, the significantly increased input length also introduces additional computational latency. In this part, we conduct experiments to measure the latency of two strategies.

Specifically, we rerank the top-100 passages retrieved by BM25 based on the sliding window strategy and full ranking strategy on three datasets—Signal, DL19, and Covid—which cover different passage lengths. In addition, as current search engines primarily display the top-10 search results, it is unnecessary to generate the whole list of passage IDs for a real-time application. Thus, we also evaluate the latency of outputting only the top-10 passage IDs based on both strategies, which can be implemented by setting the model's maximum number of output tokens. Note that in the sliding window process (with a window size of 20 and a step size of 10), only outputting 10 passage IDs at each step still ensures the final top-10 ranked IDs.

We measure the latency on a 40GB Nvidia A100 GPU and average across all queries within each dataset. For GPT-4o-mini and GPT-4o, the latency is measured through API calls. Note that, since both RankMistral100 and RankMistral20 are fine-tuned on Mistral-7B-Instruct-v0.3, we do not compare their efficiency separately.

The results are presented in Figure 3. The following observations can be drawn from the figure:

(1) Ranking latency of full ranking strategy (light orange bar) is much smaller than sliding window strategy (light green bar) across all long-context LLMs. For example, on the Signal dataset, the full ranking strategy achieves an efficiency improvement of approximately 42% on Qwen-2.5 and 65% on GPT-4o compared to the sliding window strategy. This demonstrates that the full ranking strategy is more efficient than the sliding window strategy.

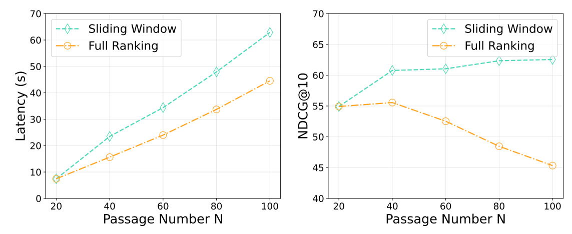
(2) When the models are restricted to output only the top-10 passage IDs, both strategies (dark green bar and dark orange bar) demonstrate significant latency reduction, indicating that the number of decoded tokens greatly impacts the overall latency. Furthermore, the latency gap between the two strategies becomes even more pronounced. For example, on the Signal dataset, the sliding window strategy takes 29.9 seconds, while the full ranking strategy only takes 3.8 seconds, resulting in

an approximate 8x speed-up. This highlights the efficiency advantage of the full ranking strategy in real-world search engines, where only the top 10 results need to be displayed after a user submits a query. Besides, the lower latency of GPT-4o and GPT-4o-mini compared to the other open-source models may be due to differences in inference resources, such as hardware acceleration and model optimization.

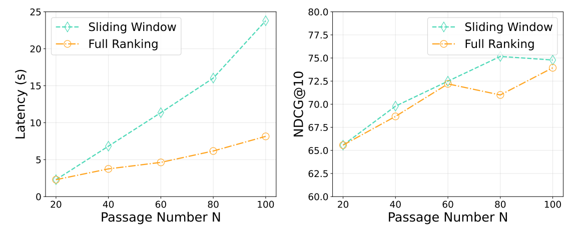
4.4 Impact of Passage Number N

Efficiency and Effectiveness Analysis In this part, we further compare the efficiency and effectiveness of the full ranking strategy and the sliding window strategy across different numbers of passages N. We choose different values of $N \in \{20, 40, 60, 80, 100\}$ and conduct the experiments on the DL19 dataset based on the open-source model Mistral-7B-instruct-v0.3 and the proprietary model GPT-4o, respectively. The results are shown in Figure 4. From the results, we have the following observations: (1) For different N, the latency of the full ranking strategy consistently remains lower than that of the sliding window strategy, with the gap becoming more noticeable as N increases. (2) As N increases, the latency of full ranking exhibits an almost linear growth, which benefits from the optimization within LLMs for handling long contexts. (3) Across various values of N, the sliding window strategy demonstrates better effectiveness compared to the full ranking strategy, similar to the phenomenon observed in Table 1. Additionally, due to GPT-4o's strong ability to model long contexts, the effectiveness gap between the two strategies is relatively smaller compared to Mistral-7B-Instruct-v0.3.

Generalization of RankMistral$_{100}$ As we mentioned in Section 3, our full ranking model is trained to rank 100 passages at a time. However, it remains unclear whether it can generalize to other numbers of passages (N). In this section, we selected different values of $N \in \{20, 40, 60, 80, 100\}$ and tested the performance of RankMistral$_{100}$ based on full ranking strategy, comparing it with RankMistral$_{20}$ using sliding window strategy. Both models are fine-tuned using our importance-aware loss L_{ia} with labels generated by GPT-4o. We conducted experiments on DL19, DL20, and BEIR, and show the results in Table 2. From the results, it can be seen that RankMistral$_{100}$ performs better than RankMistral$_{20}$ across different N values. This indicates that even though RankMistral$_{100}$ is trained on passage lists of length 100, it can still generalize to variable-length ranking tasks. The comprehensive results are presented in Table 5.



(a) Results of Mistral-7B-instruct-v0.3.



(b) Results of GPT-4o.

Figure 4: Comparison of sliding window strategy and full ranking strategy on DL19 dataset based on Mistral-7B-instruct-v0.3 and GPT-4o, respectively.

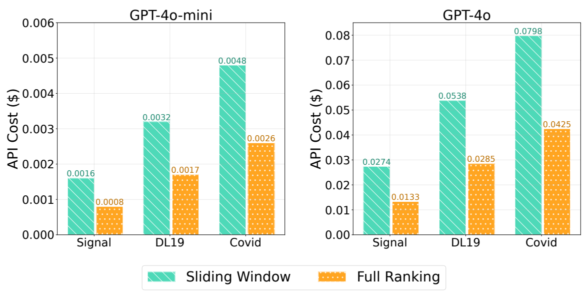


Figure 5: The comparison of API cost per query of the sliding windows strategy and the full ranking strategy when ranking top-100 retrieved passages.

4.5 API Cost Comparison

As we mentioned in Section 1, full ranking avoids the inference of redundant passages between adjacent windows in the sliding window strategy, thereby reducing the API cost for LLMs. In this section, we investigate the API cost savings of the full ranking strategy compared to the sliding window strategy. We selected the Signal, DL19, and Covid datasets, and used GPT-4o-mini and GPT-4o as the inference models. We rerank the BM25-retrieved top-100 passages, and the API costs are calculated based on the number of input and output tokens, using OpenAI's official price. As shown

N	Model	DL19	DL20	BEIR Avg
20	RankMistral$_{20}$	63.47	59.56	48.45
	RankMistral$_{100}$	**65.82**	**62.03**	**49.87**
40	RankMistral$_{20}$	65.89	63.88	50.07
	RankMistral$_{100}$	**67.77**	**66.01**	**51.88**
60	RankMistral$_{20}$	70.34	64.82	50.66
	RankMistral$_{100}$	**70.97**	**68.74**	**52.00**
80	RankMistral$_{20}$	70.28	68.29	51.13
	RankMistral$_{100}$	**71.36**	**70.22**	**52.14**
100	RankMistral$_{20}$	70.34	69.58	51.85
	RankMistral$_{100}$	**72.55**	**71.29**	**52.40**

Table 2: Results (NDCG@10) of RankMistral$_{100}$ and RankMistral$_{20}$ under different passage number N.

in Figure 5, the full ranking strategy reduces API costs by about 50% compared to the sliding window strategy, which proves its cost-efficiency. The differences in API costs across datasets are mainly due to varying passage lengths (longer passages leading to higher costs).

5 Related Work

5.1 LLMs for Passage Ranking

The application of LLMs into information retrieval (Zhu et al., 2023, 2024) has sparked significant interest, leading to numerous studies on utilizing LLMs for passage ranking tasks. Existing approaches leveraging LLMs for passage ranking can be classified into three categories based on their ranking strategies: pointwise, pairwise, and listwise methods. Pointwise methods (Liang et al., 2022; Sachan et al., 2022; Zhuang et al., 2023a; Liu et al., 2024b) assess the relevance between a query and each passage independently. Pairwise methods (Qin et al., 2023; Luo et al., 2024) involve comparing two passages at a time to determine which one is more relevant to the query. Listwise methods (Sun et al., 2023; Pradeep et al., 2023a; Reddy et al., 2024; Liu et al., 2024a) directly rank a list of passages. By taking multiple passages as input, listwise methods perform multi-passage comparisons and show promising ranking performance. Given the promising performance of listwise ranking, there has been a growing interest in exploring this approach. For example, Sun et al. (2023) introduce a prompt-based framework that utilizes ChatGPT to rank passages in a zero-shot manner. Pradeep et al. (2023a) and Pradeep et al. (2023b) further propose to distill the strong ranking ability of ChatGPT or GPT-4 into moderate-size LLMs. Liu et al. (2025) propose a novel collaborative ranking framework CoRanking, which strategically combines a small listwise reranker and a large listwise reranker for efficient and effective ranking.

Due to the limited input length of LLMs, existing listwise methods mainly take a subset of passages as input and apply a sliding window strategy to rank passages from back to front. While this approach yields promising performance, it still suffers from efficiency and cost problems as mentioned in Section 1. Long-context LLMs, with the ability to process significantly longer inputs, present new opportunities for listwise passage ranking. However, the academic community has yet to explore the application of long-context LLMs in passage ranking. In this paper, we make the first attempt to investigate this direction.

5.2 Long Context Large Language Models

Recently, there has been significant progress in developing LLMs that can process longer input sequences. The context window sizes of LLMs have grown from 1024 tokens in GPT-2 (Radford et al., 2019) to 4096 in LLaMA 2 (Touvron et al., 2023). To address the computational challenges of longer contexts, methods like efficient attention mechanisms (Ding et al., 2023; Zhang et al., 2024; Mohtashami and Jaggi, 2023) and positional interpolation (Chen et al., 2023) have been proposed. For example, Landmark attention (Mohtashami and Jaggi, 2023) extends LLaMA 7B's context length from 4K to 32K by using "landmark tokens" to represent context blocks and fine-tuning attention to select relevant tokens. Building upon these techniques, many long-context LLMs have been developed and released, such as Mistral-7B-instruct-v0.3 (32k) (Jiang et al., 2023), Qwen2.5-7B-Instruct (128k) (Yang et al., 2024) and GPT-4o (128k). Although some previous studies (Xu et al., 2024; Bai et al., 2024) have discussed the impact of long-context LLMs on retrieval, less attention has been paid to ranking tasks. In this work, we intend to provide a comprehensive discussion on the potential benefits and challenges of long-context LLMs for ranking tasks.

6 Conclusion

In this paper, we conduct a comprehensive study on passage ranking with long-context LLMs. Our experiments demonstrate that the full ranking strategy is more efficient than the sliding window strategy.

After fine-tuning, the full ranking model also outperforms the sliding window model in ranking effectiveness. For fine-tuning the full ranking model, we propose a multi-pass sliding window approach and an importance-aware learning objective for label generation and model optimization. Experiments demonstrate the effectiveness of our method.

Limitations

We acknowledge some limitations in our work. Firstly, due to our limited resources, we cannot experiment with larger open-source long-context LLMs, such as those with 30B or even 70B parameters. Investigating the impact of model size on the effectiveness and efficiency of the full ranking strategy would be an interesting direction for future research. Secondly, the efficiency advantage of the full ranking strategy is attributed to the existing long-context LLMs. We did not design a specialized long-context LLM for the ranking task. However, we consider this a promising direction and will treat it as future work.

Acknowledgments

Zhicheng Dou is the corresponding author. This work was supported by Beijing Municipal Science and Technology Project No. Z231100010323009, National Natural Science Foundation of China No. 62272467, Beijing Natural Science Foundation No. L233008, the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China No.22XNKJ34 and No. 23XNH027. The work was partially done at the Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *ACL (1)*, pages 3119–3137. Association for Computational Linguistics.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *CoRR*, abs/2306.15595.

Yiqun Chen, Qi Liu, Yi Zhang, Weiwei Sun, Daiting Shi, Jiaxin Mao, and Dawei Yin. 2024. Tourrank: Utilizing large language models for documents ranking with a tournament-inspired strategy. *CoRR*, abs/2406.11678.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020a. Overview of the TREC 2020 deep learning track. In *TREC*, volume 1266 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020b. Overview of the TREC 2019 deep learning track. *CoRR*, abs/2003.07820.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1, 000, 000, 000 tokens. *CoRR*, abs/2307.02486.

Neel Jain, Ping-yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Neftune: Noisy embeddings improve instruction finetuning. In *ICLR*. OpenReview.net.

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. *CoRR*, abs/2310.06825.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yüksekgönül, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic evaluation of language models. *CoRR*, abs/2211.09110.

Qi Liu, Bo Wang, Nan Wang, and Jiaxin Mao. 2024a. Leveraging passage embeddings for efficient listwise reranking with large language models. *CoRR*, abs/2406.14848.

Wenhan Liu, Xinyu Ma, Yutao Zhu, Lixin Su, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. 2025. Coranking: Collaborative ranking with small and large ranking agents. *CoRR*, abs/2503.23427.

Wenhan Liu, Yutao Zhu, and Zhicheng Dou. 2024b. Demorank: Selecting effective demonstrations for large language models in ranking task. *CoRR*, abs/2406.16332.

Jian Luo, Xuanang Chen, Ben He, and Le Sun. 2024. Prp-graph: Pairwise ranking prompting to llms with graph aggregation for effective text re-ranking. In *ACL (1)*, pages 5766–5776. Association for Computational Linguistics.

Amirkeivan Mohtashami and Martin Jaggi. 2023. Landmark attention: Random-access infinite context length for transformers. *CoRR*, abs/2305.16300.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Rodrigo Frassetto Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *CoRR*, abs/1901.04085.

Rodrigo Frassetto 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, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 708–718. Association for Computational Linguistics.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023a. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *CoRR*, abs/2309.15088.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023b. Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze! *CoRR*, abs/2312.02724.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2023. Large language models are effective text rankers with pairwise ranking prompting. *CoRR*, abs/2306.17563.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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. *CoRR*, abs/2406.15657.

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 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3781–3797. Association for Computational Linguistics.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 14918–14937. Association for Computational Linguistics.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS Datasets and Benchmarks*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Retrieval meets long context large language models. In *ICLR*. OpenReview.net.

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, Jianxin Yang, 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, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *CoRR*, abs/2407.10671.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2024. Soaring from 4k to 400k: Extending llm's context with activation beacon. *CoRR*, abs/2401.03462.

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. *CoRR*, abs/2308.07107.

Yutao Zhu, Peitian Zhang, Chenghao Zhang, Yifei Chen, Binyu Xie, Zheng Liu, Ji-Rong Wen, and Zhicheng Dou. 2024. INTERS: unlocking the power of large language models in search with instruction tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2782–2809. Association for Computational Linguistics.

Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Bendersky. 2023a. Beyond yes and no: Improving zero-shot LLM rankers via scoring fine-grained relevance labels. *CoRR*, abs/2310.14122.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023b. Rankt5: Fine-tuning T5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 2308–2313. ACM.

A Listwise Ranking Prompt

The listwise ranking prompt used in this work is shown below:

Prompt: Listwise Ranking Prompt

You are RankLLM, an intelligent assistant that can rank passages based on their relevancy to the query.
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]` {passage 1}
`[2]` {passage 2}
...
`[{num}]` {passage {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.

B Training Configuration

Our backbone model Mistral-7B-Instruct-v0.3 for supervised-fine-tuning is available at `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`. We apply the prompt shown in Figure A to construct the training input. We randomly sample 1k queries from MS MARCO training set to generate the teacher labels. During our experiments, we find that using more queries for training does not yield better results (see Figure 6). Both RankMistral$_{20}$ and RankMistral$_{100}$ are finetuned for 4 epochs with a learning rate of 5e-6 and batch size of 1. The hyper-parameter α used in Equation 2 is set as 1. Following RankZephyr Pradeep et al. (2023b), we apply noisy embeddings (Jain et al., 2024) and bfloat16 precision during model training. We conduct all the experiments on 4 A100-40G GPUs.

C The Impact of Passage Order and Ranking Number

Previous studies have revealed that the initial passage order and ranking number have an impact on the ranking performance. In this section, we explore the ranking performance of our RankMistral$_{100}$ distilled from GPT-4o on the TREC DL19 dataset under different initial order and ranking number settings to gain further insights into full ranking.

Regarding the initial passage order, we experimented with two different configurations: random order and reverse order. The results in Figure 3 show that the initial order of passages impacts the ranking performance, suggesting that a good initial order is important for listwise ranking. This is consistent with the conclusions from previous work (Sun et al., 2023; Chen et al., 2024). There are also some training techniques aimed at enhancing the robustness of listwise rankers to the initial order, such as shuffling the order of training passages during training; however, this is not the focus of our paper. We also observe that increasing the ranking number improves the ranking performance. However, when the ranking number reaches 3 or 4, the performance gains start to converge.

Method	NDCG@1	NDCG@5	NDCG@10
BM25	54.26	52.78	50.58
RankMistral$_{100}$	72.87	74.19	72.55
Initial passage order			
Random order	70.54	72.95	69.20
Reverse order	71.32	74.09	68.36
Number of ranking			
Rank 2 times	74.42	76.46	73.92
Rank 3 times	74.42	77.53	74.30
Rank 4 times	**74.42**	**77.61**	**74.50**

Table 3: The impact of initial passage order and number of ranking on RankMistral$_{100}$ on TREC DL19.

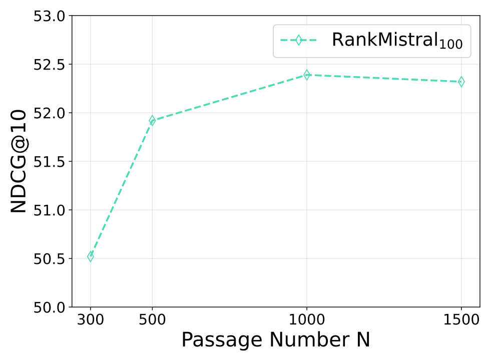


Figure 6: The results (NDCG@10) of RankMistral$_{100}$ on BEIR Avg using different numbers training queries. The figure shows that increasing the number of training queries from 1000 to 1500 does not improve performance on BEIR Avg.

Model	$USD / 1K Input Tokens	$USD / 1K Output Tokens
GPT-4o-mini-2024-07-18	0.00015	0.00060
GPT-4o-2024-08-06	0.0025	0.0100

Table 4: The price of GPT-4o-mini-2024-07-18 and GPT-4o-2024-08-06.

N	Model	DL19	DL20	Covid	DBPedia	SciFact	NFCorpus	Signal	Robust04	Touche	News	Avg.
20	RankMistral$_{20}$	63.47	59.56	69.98	37.78	74.42	36.46	32.29	48.49	40.55	**47.61**	48.45
	RankMistral$_{100}$	**65.82**	**62.03**	**70.86**	**38.73**	**75.64**	**37.11**	**35.33**	**50.67**	**43.43**	47.17	**49.87**
40	RankMistral$_{20}$	65.89	63.88	75.91	40.75	75.40	36.91	33.35	52.10	39.00	47.06	50.07
	RankMistral$_{100}$	**67.77**	**66.01**	**77.46**	**41.79**	**76.79**	**38.25**	**35.13**	**53.27**	**43.05**	**49.31**	**51.88**
60	RankMistral$_{20}$	70.34	64.82	78.04	41.43	74.79	37.97	33.38	54.31	37.02	48.30	50.66
	RankMistral$_{100}$	**70.97**	**68.74**	**79.96**	**41.92**	**76.70**	**38.66**	**34.18**	**55.93**	**38.45**	**50.18**	**52.00**
80	RankMistral$_{20}$	70.28	68.29	80.15	42.04	75.91	37.86	**33.77**	54.65	35.09	49.53	51.13
	RankMistral$_{100}$	**71.36**	**70.22**	**81.25**	**42.38**	**76.35**	**38.95**	33.47	**56.67**	**36.96**	**51.10**	**52.14**
100	RankMistral$_{20}$	70.34	69.58	80.86	42.52	75.82	38.38	33.90	54.69	**37.18**	**51.42**	51.85
	RankMistral$_{100}$	**72.55**	**71.29**	**82.24**	**43.54**	**77.04**	**39.14**	**33.99**	**57.91**	34.63	50.59	**52.40**

Table 5: Results (NDCG@10) on TREC and BEIR under different passage number N. Best performing models are marked in bold. All the models are finetuned with labels generated by GPT-4o.

D API Cost

In Table 4, we present the price for two different teacher models we used in this paper. We use 1k query to generate the teacher label for different models. For fine-tuning the sliding window model RankMistral$_{20}$, we re-rank the top-20 passages retrieved by BM25 using the teacher model, costing about $1.7 for GPT-4o-mini and $29 for GPT-4o. For the full ranking model, we use the multi-pass sliding window strategy to obtain the full ranked list, with costs of around $15.7 for GPT-4o-mini and $261 for GPT-4o.

E Baselines Details

The baselines we used for comparison are:

• **monoBERT** (Nogueira and Cho, 2019): A BERT-large based cross-encoder re-ranker, fine-tuned using the MS MARCO dataset (Nguyen et al., 2016).

• **monoT5** (Nogueira et al., 2020): A re-ranker using a sequence-to-sequence model with T5 to determine relevance scores.

• **RankT5** (Zhuang et al., 2023b): A T5-based pointwise re-ranker trained using ranking loss.

• **RankVicuna** (Pradeep et al., 2023a): A listwise reranker fine-tuned from GPT-3.5 generated ranked list.

• **RankZephyr** (Pradeep et al., 2023b): A listwise reranker distilled from GPT-3.5 and GPT-4 with a two-stage training process.

Method	Latency (s/query)
Sliding Window	85.37
Full Ranking	**61.91**
Sliding Window (Output Top-10 ID)	42.93
Full Ranking (Output Top-10 ID)	**6.90**

Table 6: The efficiency comparison of Qwen-2.5 (14B).

F Experiments with Larger Open-source LLMs

In this section, we compare the full ranking strategy and sliding window strategy in a zero-shot setting with larger open-source LLMs. We choose Qwen2.5-14B-Instruct (denoted as Qwen-2.5 (7B)) as our LLM and show the comparison of ranking effectiveness in Table 7. We also report the results of Qwen2.5-7B-Instruct for comparison. From the table, we can see that full ranking with Qwen-2.5 (14B) underperforms the sliding window strategy, which is the same as with Qwen-2.5 (7B). Besides, Qwen-2.5 (14B) outperforms Qwen-2.5 (7B) on all datasets, indicating that a larger model size results in higher performance.

We also utilize Qwen-2.5 (14B) to compare the efficiency of the full ranking and sliding window strategies on the DL19 dataset. The comparison results are shown in Table 6 (“Output Top-10 ID” indicates that the LLM only generates the top-10 ranked passage IDs). From the table, we can see that full ranking is more efficient than the sliding window strategy, especially when only outputting

Models	Strategy	DL19	DL20	Covid	DBPedia	SciFact	NFCorpus	Signal	Robust04	Touche	News	Avg.
Qwen-2.5 (7B)	Sliding	68.34	64.89	79.76	40.87	71.32	37.97	30.66	52.87	32.02	49.49	49.37
Qwen-2.5 (7B)	Full	57.44	54.42	65.06	35.57	61.49	33.75	27.35	37.60	26.91	37.55	40.66
Qwen-2.5 (14B)	Sliding	**70.57**	**66.80**	**80.84**	**43.42**	**73.53**	**39.06**	**32.78**	**55.42**	**38.08**	**49.94**	**51.63**
Qwen-2.5 (14B)	Full	59.39	58.26	67.29	37.56	69.69	35.65	28.64	41.00	36.14	37.72	44.21

Table 7: The ranking performance comparsion of Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct.

Teacher	Sampling Strategy	DL19	DL20	Covid	DBPedia	SciFact	NFCorpus	Signal	Robust04	Touche	News	Avg.
GPT-4o-mini	Top-20	**69.20**	**68.66**	79.72	**43.54**	**73.93**	37.49	**32.74**	**54.95**	**32.69**	50.21	**50.66**
	Sample 20	69.13	67.49	**80.44**	42.06	73.04	**37.97**	32.26	54.19	31.50	**50.30**	50.22
GPT-4o	Top-20	**71.14**	**68.49**	80.80	**42.53**	**76.15**	**38.54**	**33.12**	**55.63**	**31.62**	**50.86**	**51.16**
	Sample 20	69.55	65.78	**81.71**	41.21	75.35	37.71	31.70	55.57	30.71	48.44	50.30

Table 8: The performance (NDCG@10) of RankMistral$_{20}$ on TREC and BEIR based on different sampling strategy. Top-20 refers to the top-20 passages retrieved by BM25, while sample 20 indicates a random sampling of 20 passages from the top-100 passages retrieved by BM25.

top-10 passage IDs. This further demonstrates that full ranking with long-context models is more efficient compared to the sliding window strategy.

Models	Strategy	DL19	DL20	Covid	DBPedia	SciFact	NFCorpus	Signal	Robust04	Touche	News	Avg.
RankQwen$_{20}$	Sliding	**72.79**	67.96	81.18	42.33	75.79	**38.53**	31.10	**56.97**	30.93	49.06	50.74
RankQwen$_{100}$	Full	71.45	**68.74**	**82.64**	**42.84**	**76.30**	38.23	**32.45**	55.93	**36.00**	**49.51**	**51.74**

Table 9: The performance comparison between the finetuned sliding window model and the full ranking model (denoted as RankQwen$_{20}$ and RankQwen$_{100}$, respectively) when using Qwen2.5-7B-Instruct as backbone model. The training labels for both models are generated by GPT-4o. The results indicate that RankQwen$_{100}$ outperforms RankQwen$_{20}$ on most datasets and achieves an average of 1-point improvement on BEIR Avg.