Leveraging Passage Embeddings for Efficient Listwise Reranking with Large Language Models

Anonymous ACL submission

Abstract

Recent studies have demonstrated the effectiveness of using large language language models (LLMs) in passage ranking. The listwise approaches, such as RankGPT, have become new state-of-the-art in this task. However, the efficiency of RankGPT models is limited by the maximum context length and relatively high latency of LLM inference. To address these issues, in this paper, we propose PE-Rank, leveraging the single passage embedding as a good context compression for efficient listwise passage reranking. By treating each passage as a special token, we can directly input passage embeddings into LLMs, thereby reducing input length. Additionally, we introduce an inference method that dynamically constrains the decoding space to these special tokens, accelerating the decoding process. For adapting the model to reranking, we employ listwise learning to rank loss for training. Evaluation results on multiple benchmarks demonstrate that PE-Rank significantly improves efficiency in both prefilling and decoding, while maintaining competitive ranking effectiveness.

1 Introduction

Passage ranking, which aims to rank each passage in a large corpus according to its relevance to the user's information need expressed in a short query, is an important task in IR and NLP and plays a crucial role in many applications such as web search and retrieval-augmented generation. To achieve both effectiveness and efficiency, current mainstream approaches usually follow a two-stage paradigm known as "*retrieval-then-rerank*", which involves efficiently retrieving a set of candidates first, and further reranking them with a reranker to boost the effectiveness (Nogueira et al., 2019).

In the first retrieval stage, dense retrieval models based on a bi-encoder architecture are widely used (Karpukhin et al., 2020). Trained on large-scale text pairs through contrastive learning, these models can encode text information into a low-dimensional dense embedding and capture semantic relevance using vector similarity.

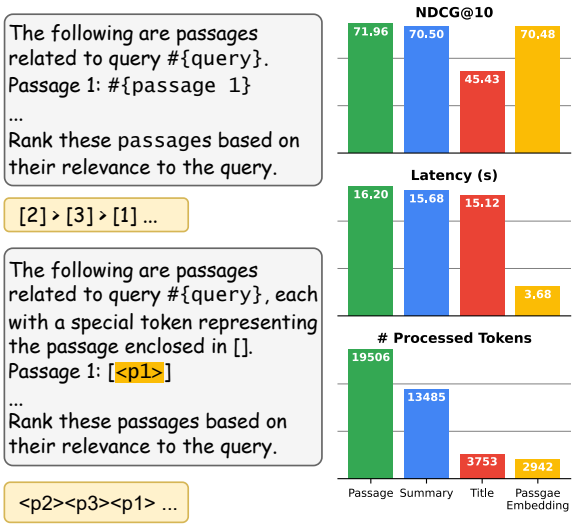

Figure 1: Comparison between RankGPT (upper) and PE-Rank (lower). RankGPT takes the whole passages as input and outputs ordered numbers, while PE-Rank takes a list of special tokens as both input and output. On the right side, we show the reranking results on DL19 using different forms of inputs.

In the second reranking stage, we can employ more sophisticated models for better ranking performance. A common reranking model is a supervised model based on the cross-encoder design (Nogueira et al., 2019). With the emergence of LLMs, such as GPT-4 (OpenAI, 2024), a series of studies have tried to leverage LLMs' text comprehension and reasoning abilities for zero-shot reranking. Typically, there are three main prompting approaches: *pointwise* (Liang et al., 2022; Sachan et al., 2022), *pairwise* (Qin et al., 2023), and *listwise* (Sun et al., 2023; Pradeep et al., 2023a). Among these methods, listwise approaches like RankGPT (Sun et al., 2023) are regarded as the most effective, achieving state-of-the-art performance by directly producing

a final ranking list for multiple passages, rather than merely assessing the relevance of a single passage or the relative position between two passages.

While the listwise approaches demonstrate potential in the reranking task, they are limited by two challenges. Firstly, LLMs are limited by context length and cannot rank multiple passages simultaneously, necessitating techniques such as a sliding window strategy to complete the ranking process (Sun et al., 2023). Secondly, incorporating entire passages into prompts significantly increases inference costs, resulting in high latency, which is untenable in the ranking scenario.

To tackle these issues, it is imperative to compress listwise reranking prompts. Some context compression methods have been proposed for LLMs and can be categorized into two types: compressing the context into dense memory slots (Mu et al., 2024; Chevalier et al., 2023; Ge et al., 2023) and directly editing the input contexts (Jiang et al., 2023b). Nonetheless, existing methods exhibit relatively low compression rates and usually only compress a single passage, rendering them inadequate for ranking tasks.

For, we first highlight that in the "*retrieval-then-rerank*" pipeline, dense retrieval models have been trained as effective text compressors with their embedding capable of representing nearly as much information as the original text (Morris et al., 2023). From this perspective, in the paper, we propose a novel and efficient listwise passage reranking method named **PE-Rank**, leveraging the single embedding of the passage as the compressed representation. Specifically, we obtain the passage embedding from a dense retrieval model and regard it as a special token of the LLM to replace the original text as input. To align the embedding space of the retrieval model and the input embedding space of the LLM, we use a projector as a bridge between the two models, which is inspired by previous work about modality alignment (Liu et al., 2024).

To adapt PE-Rank to ranking tasks, we propose novel inference and training methods. For accurate and efficient inference, we propose a "Dynamic-Constrained Decoding" strategy that dynamically changes the decoding spaces to a set of special tokens that represent the rest of the passages to be ranked. We employ two-stage training, first training the projector for modality alignment, then training both the projector and LLM for ranking tasks using listwise learning to rank loss.

We evaluate PE-Rank on popular retrieval benchmarks TREC DL and BEIR. Experimental results demonstrate that PE-Rank achieves comparable ranking performance to uncompressed methods while significantly improving inference efficiency. Notably, when reranking top 100 candidates retrieval by BM25 on DL19, NDCG@10 of PE-Rank is only reduced by less than 2% compared to the uncompressed method under the same settings while reducing the latency by a factor of 4.5.

In summary, the main contributions of this paper are as follows:

- We propose a novel efficient listwise reranking method, PE-Rank, first using passage embeddings for context compression in ranking.
- We evaluate PE-Rank on multiple benchmarks and show its competitive ranking performance and significant efficiency advantages.

2 Methodology

2.1 Overview

The overview architecture of PE-Rank is shown in Figure 2, we introduce the model under the two-stage ranking paradigm.

Specifically, we first use the dense retrieval model to pre-encode the corpus into a vector index. Given a query q, we use the same encoder to encode it into an embedding and retrieve several most relevant candidate passages $\mathcal{P}_{cand} = [p_1, ..., p_n]$ and their embeddings $\boldsymbol{e}_{p_1}, ..., \boldsymbol{e}_{p_n}$. Here vector similarity is used as the relevance score between query and passages.

In the reranking stage, our key idea is to take the embeddings from the previous stage as a good context compression of passages. Therefore, we propose replacing the original passage with the single embedding representation as the input of LLMs. However, there are dimensional and distribution differences between the passage embeddings and LLM's token embeddings, which require us to bridge the gap between two spaces with a learned mapping function. Taking inspiration from previous work on aligning two modalities (Liu et al., 2024), we introduce a two-layer MLP, denoted as $\mathbf{E}_M$, as the mapping function. Here we treat these transformed embeddings $\mathbf{E}_M(\boldsymbol{e}_{p_i})$ as the embeddings of additional out-of-vocabulary special tokens, where one passage is represented as one special token, for example <p1> represents p_1.

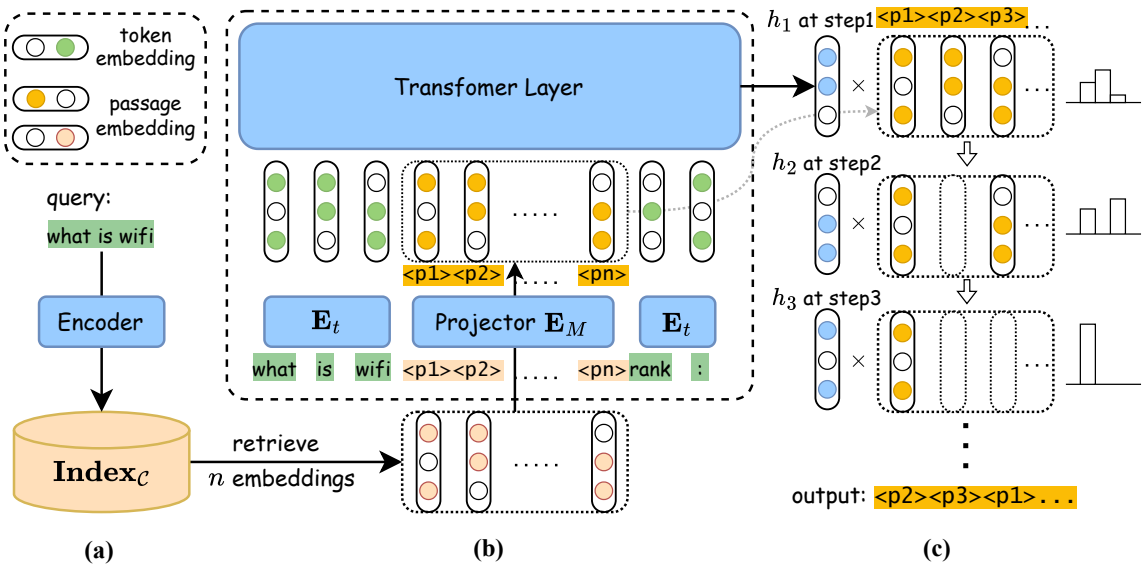

Figure 2: Overview of PE-Rank under a two-stage ranking paradigm. **(a)** is retrieval stage, retrieve n passage embeddings; **(b)** is the forward pass procedure of LLM; **(c)** shows the listwise decoding process.

Furthermore, by taking the instruction I and query q as normal tokens and then concatenating the token embeddings and transformed passage embeddings, we can define the simplified input embeddings of LLM at the first generation step:

$$\mathbf{E}_{\text{In}}^{(1)} = \mathbf{E}_t(I \oplus q) \oplus \mathbf{E}_M(\boldsymbol{e}_{p_1}) \cdots \oplus \mathbf{E}_M(\boldsymbol{e}_{p_n}), \quad (1)$$

where $\mathbf{E}_t$ is the token embedding layer of LLM. The complete prompts are listed in Appendix F. In the next section, we will introduce how to output the ranking list in detail.

It should be pointed out that although we describe PE-Rank in the background of two-stage ranking, it can be applied separately for reranking, simply using the encoder as a text compressor by encoding passages on the fly.

Algorithm 1: DC Decoding

Input : Candidates $\mathcal{P}_{cand} = [p_1, ..., p_n]$, Intial Input Embeddings $\mathbf{E}_{\text{In}}^{(1)}$

Output : Ranking List $\hat{\mathcal{P}}_{rank} = [\hat{p}_1, ..., \hat{p}_n]$

$\hat{\mathcal{P}}_{rank} \leftarrow \emptyset$

for $i \leftarrow 1$ **to** n **do**

  $\boldsymbol{h}_i \leftarrow \text{Transformer}(\mathbf{E}_{\text{In}}^{(i)})$

  $\hat{p}_i \leftarrow \arg\max_{p \in \mathcal{P}_{cand}} (\boldsymbol{h}_i^T \cdot \mathbf{E}_M(\boldsymbol{e}_p))$

  $\mathbf{E}_{\text{In}}^{(i+1)} \leftarrow \mathbf{E}_{\text{In}}^{(i)} \oplus \mathbf{E}_M(\boldsymbol{e}_{\hat{p}_i})$

  $\mathcal{P}_{cand}$.remove($\hat{p}_i$)

  $\hat{\mathcal{P}}_{rank}$.append($\hat{p}_i$)

end

return $\hat{\mathcal{P}}_{rank}$

2.2 Inference

During inference, listwise rerankers aim to output a ranking list directly. For LLM-based listwise approaches, we usually generate the ranking list autoregressively. In previous work, LLMs are prompted to generate a string that could be parsed into a ranking list, such as "[2] > [3] > [1]..." (Sun et al., 2023; Pradeep et al., 2023a). However, in early experiments, we found that generating a string representing ranking may be difficult and slow, as LLM may output in the wrong format or output useless content, such as explanation.

To address this issue, we propose a "Dynamic-Constrained Decoding" (DC Decoding for short) strategy in Algorithm 1. During decoding, we dynamically change the decoding spaces according to the rest of the passages that need to be ranked, treating the embedding representation of those passages as a special set of tokens. At each generation step, we no longer output a normal numerical token but instead constrain the decoding space only in these special tokens, to perform accurate ranking. Therefore, we can directly output a list of tokens that represent the ranking of passages, such as "<p2><p3><p1>...". Furthermore, as the decoding space and the number of generated tokens are much smaller than the original vocabulary space, inference will be accelerated.

For example, as shown in Figure 2 (c), we first obtain the hidden state $\boldsymbol{h}_1$ from LLM in the first decoding step and calculate the output probability distribution with all the passages embeddings $\mathbf{E}_M(\boldsymbol{e}_{p_1}), ..., \mathbf{E}_M(\boldsymbol{e}_{p_n})$, then take the p_2 with the highest probability as the top-1 passage in the result. In the second decoding step, we append $\mathbf{E}_M(\boldsymbol{e}_{p_2})$ to the input embeddings of LLM at last, remove it from the decoding space, and use the hidden state $\boldsymbol{h}_2$ in the second step to get the next output. By repeating this process, we obtain the final ranking.

We use the greedy search algorithm in the actual inference process. It should be pointed out that when generating the next special token, the model relies on the previously predicted results rather than the ground truth.

2.3 Training

During training, we aim to address two challenges: aligning disparate embedding spaces and adapting the model for ranking. Consequently, we divide the training into two stages: (1) the alignment stage, which aligns the output space of the dense retrieval model with the token embedding space of the LLM, and (2) the learning-to-rank stage, which enables the model to acquire knowledge about ranking.

Alignment stage At this stage, our objective is to ensure that the passage embeddings produced by the dense retrieval model are comprehensible to the large language model and effectively represent the original text information. To achieve this, we design a text reconstruction task for training. Given a piece of text t, it is first encoded into an embedding and passed through the MLP. Taking the transformed embedding as part of the input, the LLM is prompted to reconstruct the original text based on the embedding. The simplified input of LLM can be formalized as:

$$\mathbf{E}_{\text{In-Align}} = \mathbf{E}_t(I) \oplus \mathbf{E}_M(\boldsymbol{e}_t), \tag{2}$$

We employ language modeling loss for training:

$$\mathcal{L}_{\text{Align}} = -\sum_{i=1} \log P_\theta(t_i | \mathbf{E}_{\text{In-Align}} \oplus \mathbf{E}_t(t_{<i})). \tag{3}$$

Note that we freeze the encoder and the LLM and only fine-tune the parameters of MLP, that is, we only learn the mapping between two different embedding spaces, without changing themselves.

Learning-to-rank stage Previous listwise approaches employed supervised fine-tuning (SFT) paradigms for training (Pradeep et al., 2023a,b). By distilling from existing reranking models, the LLMs acquire ranking knowledge. However, the dynamic nature of the decoding space renders standard SFT inapplicable in this context.

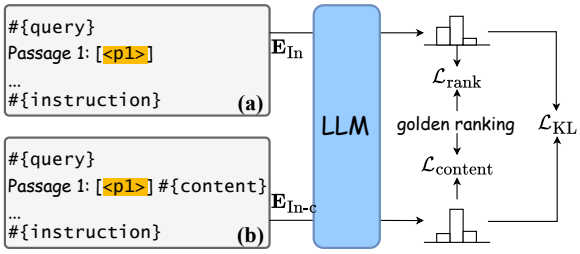

Figure 3: Illustration of two types of training data and the learning-to-rank training process.

Therefore, we propose that the decoding process can be viewed as a sequential ranking learning process: at each step, we provide the previously decoded rankings and maximize the probability of generating the next most relevant passage. Formally, if given a query q and the golden ranking list $[p_1, ..., p_n]$, at step i, we maximize the conditional probability of p_i given q and previous $p_{<i}$:

$$\begin{aligned} P_\theta(p_i | q, p_{<i}) &= P_\theta(p_i | \mathbf{E}_{\text{In}}^{(i)}) \\ &= \frac{\exp(\boldsymbol{h}_i^T \cdot \mathbf{E}_M(\boldsymbol{e}_{p_i}))}{\sum_{j=i}^{n} \exp(\boldsymbol{h}_i^T \cdot \mathbf{E}_M(\boldsymbol{e}_{p_j}))}, \end{aligned} \tag{4}$$

where θ is the model's parameters. Considering the whole sequential process, it is equivalent to listwise learning to rank loss ListMLE (Xia et al., 2008):

$$\mathcal{L}_{\text{rank}} = -\sum_{i=1}^{n} \log P_\theta(p_i | \mathbf{E}_{\text{In}}^{(i)}). \tag{5}$$

Here we only leverage the passage embeddings for ranking, as illustrated in the prompt (a) in Figure 3. The full prompts can be found in Appendix F.

However, understanding entire passages with single embedding and utilizing them for ranking may be challenging for LLMs, which may result in difficulties when directly training with Equation (5). Therefore, we incorporate both the original text and the passage embedding into the model inputs and apply the same forward pass to compute the loss:

$$\mathcal{L}_{\text{content}} = -\sum_{i=1}^{n} \log P_\theta(p_i | \mathbf{E}_{\text{In-c}}^{(i)}), \tag{6}$$

where $\mathbf{E}_{\text{In-c}}^{(i)}$ is defined similarly as Equation (1), but includes the content as part of the input, as illustrated in the prompt (b) in Figure 3. We believe

this approach enhances the model's ability to utilize the token-level interactions between query and passage and helps transfer this ability when solely using embeddings for ranking.

Additionally, we also employ KL Divergence for distillation, which enables the model using compressed embeddings to emulate the proficiency in handling the uncompressed texts:

$$\mathcal{L}_{\text{KL}} = \sum_{i=1}^{n} D_{\text{KL}}(P_\theta(p_i|\mathbf{E}_{\text{In}}^{(i)}) \| P_\theta(p_i|\mathbf{E}_{\text{In-c}}^{(i)}). \quad (7)$$

The final loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{rank}} + \mathcal{L}_{\text{content}} + \alpha \mathcal{L}_{\text{KL}}. \quad (8)$$

Here α is set to 0.2. We fine-tune both MLP and LLM in this stage but remain encoder frozen.

It is important to note that during training, we use the golden ranking labels at each step, which differs from the inference process.

3 Experiment Setup

3.1 Evaluation Datasets

We evaluate PE-Rank on multiple retrieval benchmarks, including TREC DL (Craswell et al., 2020) and BEIR (Thakur et al., 2021). TREC DL uses the MS MARCO dataset (Bajaj et al., 2016) as the retrieval corpus and has fine-grained relevance annotations. We use the test sets of TREC DL 2019 and TREC DL 2020, which contain 43 and 54 queries respectively. BEIR contains 18 datasets from different fields with different query requirements, aiming to evaluate the generalization ability of ranking models. Following previous work (Sun et al., 2023), we conduct evaluations on 8 datasets that contain a relatively small number of queries. We use nDCG@10 as evaluation metrics.

3.2 Implementation Details

We choose Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a) as our backbone model since it has a strong instruction-following ability. For most experiments, we select one popular embedding model, i.e., Jina-Embeddings (Günther et al., 2023), which has 137M parameters and shows a strong generalization ability across different corpora. Also, we use different embedding models in the ablation study to demonstrate that our framework can adapt to other models. We will use PE-Rank$_\star$ to denote different embedding models, but for convenience, if not indicated, Jina-Embeddings is used.

As for training data, we leverage Wikipedia for alignment and MS MARCO for the learning-to-rank stage. For the latter, we use a retrieval model to obtain the top 20 candidate passages for the queries in the training set and employ a cross-encoder as the teacher model to estimate the golden ranking. More details about data construction, model selection, and implementation are listed in Appendix B and C.

During the evaluation, for each dataset, we first use a retrieval model to recall the top 100 passages for each query, and then evaluate the reranking results. For convenience, we encode the passages on the fly, allowing us to use different retrieval models to provide a more comprehensive comparison. If not otherwise specified, we use the sliding window trick to complete the whole ranking and set the window size to 20 and the step size to 10, therefore need 9 passes in total. We use one Nvidia H100 GPU to finish all evaluations.

3.3 Baselines

We select several existing methods as our basic baselines, including supervised neural reranking models monoBERT (Nogueira et al., 2019) and monoT5 (Nogueira et al., 2020) that are trained using a large amount of human annotation data, unsupervised LLM-based listwise approach RankGPT (Sun et al., 2023), as well as several listwise ranking models that are based on smaller LLMs and trained with distillation including RankVicuna (Pradeep et al., 2023a) and RankZephyr (Pradeep et al., 2023b).

For a fair comparison, we train a model using a similar paradigm as RankVicuna (Pradeep et al., 2023a) but use the Mistral-7B and the training data same as PE-Rank, denoted as **RankMistral**.

Also, we use this model to evaluate different text compression strategies and compare them with PE-Rank. Specifically, we can use different texts to replace the original passage in the inputs, denoted as **RankMistral**$_*$, where $*$ can be passage (p), summary (s), or title (t). We provide more details on baselines in Appendix D.

4 Experiment Results

4.1 Effectiveness Analysis

We first evaluate the effectiveness of PE-Rank on TREC DL and BEIR benchmarks, and present the results in Table 1. From the results, we can observe that the supervised models based on BERT

Model	*Ret.*	DL19	DL20	BEIR Avg.
BM25	-	50.58	47.96	43.42
Jina-Embedding	-	65.94	63.89	41.46
Supervised models trained with human annotation				
monoBERT	BM25	70.50	67.28	47.16
monoT5		71.83	68.89	51.36
Unspervised LLM-based listwise models				
$\text{RankGPT}_{3.5}$	BM25	65.80	62.91	49.37
RankGPT_{4}		**75.59**	**70.56**	**53.68**
LLM-based listwise models trained with distillation				
RankVicuna	BM25	66.82	65.49	-
RankZephyr		74.20	70.86	-
RankMistral		71.73	68.07	43.65
PE-Rank		70.48	63.54	47.96
RankVicuna	Jina	69.81	70.61	-
RankZephyr		69.83	75.15	-
RankMistral		71.44	73.27	42.86
PE-Rank		70.91	69.48	44.28

Table 1: Results (NDCG@10) of reranking top-100 passages on TREC DL and BEIR. ***Ret*** means the retrieval model used in first stage.

and T5 can achieve competitive ranking performance, while in the LLM-based baselines, using the strongest LLM, GPT-4, for listwise reranking can achieve state-of-the-art across all models on three datasets. As for distilled models, RankZephyr also shows promising ranking effectiveness, and we attribute this to using GPT-4 as the teacher model.

Comparing the proposed PE-Rank model with other baselines, we can see that: (i) without directly trained with human-annotated data, PE-Rank can approach supervised baselines' performance. (ii) Despite compressing the entire passage into a single embedding, PE-Rank still maintains comparable effectiveness to the uncompressed distilled listwise models, even surpassing them on some datasets. For example, comparing PE-Rank with RankMistral, we can find that its ranking performance on DL19 has decreased less than 2%, while the results on BEIR are even consistently higher.

It should be emphasized that when PE-Rank remains competitive, it has a significant efficiency advantage, and we will provide a detailed analysis in the next section.

4.2 Efficiency Analysis

We conduct efficiency analysis from the perspectives of consumed tokens and latency on DL19 and Covid. We select Covid as it has a relatively long passage length, while the results on DL20 are similar to those on DL19.

Number of Consumed Tokens We theoretically analyze the number of *processed* tokens in the prefill stage and *generated* tokens in the decode stage of different methods. Assume a single pass with n passages of average length L_p and instruction of length L_I, methods based on the text like RankGPT exhibit an input length of $O(L_I + nL_p)$, which increases almost proportionally with L_p. In contrast, PE-Rank shows an input length of $O(L_I + n)$ which will be unchanged when L_p increases. For RankGPT-like methods, they need to generate numbers as well as identifiers such as "[]" and may not output completely correctly, resulting in the number of generated tokens for $\Omega(mn)$. In practice $m \approx 4.5$. As for PE-Rank, by employing the DC decoding method, the number is exactly equal to n since only n unique special tokens will be output.

It is important to note that when employing the sliding window strategy, the above results must be multiplied by the number of sliding windows. However, PE-Rank, due to the compression of input length, can achieve completion with fewer sliding instances or even in a single pass, thereby further underscoring its efficiency advantages.

Table 2 displays the number of tokens consumed by different methods. The results show that, although simple text compression techniques partially reduce tokens to be processed, they may lead to significant performance degradation. Specifically, when using titles as compression on DL19, the performance is notably poor, possibly due to title misses or lack of valid information. Using summaries as input also results in performance loss, particularly on the Covid dataset. Besides, these text-based methods do not decrease the number of generated tokens. Note that the model may not output in the required format in practice, leading to fluctuations in the number of generated tokens.

In contrast, PE-Rank significantly reduces the number of tokens to be processed and generated, while minimizing the loss of ranking performance. Surprisingly, when ranking the top 20 passages on the Covid dataset, it even outperforms the approach without compression.

Latency We also analyze the reranking latency using different methods in Table 2. The results indicate that heuristic text compression techniques, such as using titles or summaries, do not significantly reduce latency. Conversely, by leveraging passage embedding as a text compression representation, PE-Rank markedly accelerates the ranking

Model	n	TREC DL19 NDCG	# Proc.	# Gen.	Latency (s)	TREC Covid NDCG	# Proc.	# Gen.	Latency (s)
RankMistral$_p$	20	**64.65**	2265.8	109.9	2.04	70.90	8190.9	110.4	2.51
RankMistral$_s$		63.03	1490.7	106.1	1.99 (×.98)	65.15	2224.2	100.2	1.92 (×.76)
RankMistral$_t$		48.62	409.5	107.2	1.93 (×.95)	66.71	829.7	110.4	1.89 (×.75)
PE-Rank		62.66	326.9	20.0	0.42 (×.21)	**72.34**	344.3	20.0	0.44 (×.18)
RankMistral$_p$	100	**71.96**	19506.2	910.2	16.20	**77.80**	71431.2	986.5	21.46
RankMistral$_s$		70.50	13485.3	881.6	15.68 (×.97)	73.85	20148.6	929.6	16.94 (×.79)
RankMistral$_t$		45.43	3753.4	865.1	15.12 (×.93)	75.40	7555.0	916.9	15.87 (×.74)
PE-Rank		70.48	2942.4	180.0	3.62 (×.22)	77.72	3098.9	180.0	3.65 (×.17)

Table 2: Efficiency analysis for reranking top n candidates retrieved by BM25 on TREC DL19 and TREC Covid. **# Proc** and **# Gen** mean the number of processed tokens in the prefill stage and generated tokens in the decode stage, respectively. For PE-Rank, we also include the time for encoding the passages on the fly. L_p of DL19 and Covid is approximately 100 and 423, respectively. The best model in each block is in bold, and the second best is underlined.

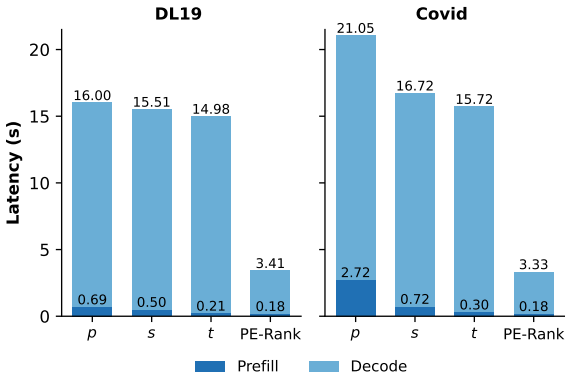

Figure 4: Latency of reranking top 100 candidates at different stages during inference.

process, achieving approximately a five-fold increase in speed across different candidate numbers and datasets, with only about 0.2 times the delay of the uncompressed method. Notably, when reranking the top 20 candidates, the ranking latency for a single query can be limited to 0.5 seconds, rendering it practical for real-world ranking scenarios.

To fully comprehend the efficiency advantages of PE-Rank, we subdivide the sources of latency into prefilling and decoding, and conduct a more detailed analysis, as shown in Figure 4. Our findings first indicate that latency predominantly arises from decoding, with prefilling contributing only minimally. On datasets with shorter passage lengths, such as DL19, PE-Rank does not demonstrate a significant efficiency advantage during the prefilling stage; instead, the advantage is primarily observed in decoding, as fewer tokens need to be output, as previously analyzed. As passage length increases, given that the input length for PE-Rank does not increase linearly, it also exhibits efficiency advantages in prefilling, as the results observed on Covid.

	DL19	DL20	Covid	News
(a) PE-Rank	**70.48**	63.54	**77.72**	47.40
(b) w/o Alignment	65.83	61.35	73.12	46.71
(c) w/o $\mathcal{L}_{\text{content}}$ & $\mathcal{L}_{\text{KL}}$	68.43	**64.42**	77.21	46.23
(d) w/o $\mathcal{L}_{\text{KL}}$	68.43	64.03	76.33	**47.42**
(e) w/o $\mathcal{L}_{\text{content}}$	66.66	60.85	75.94	47.15

Table 3: Ablation on different training strategies. We show the results of ranking top 100 candidates of BM25.

4.3 Ablation Study

Training Strategies We analyze the impact of various training strategies on PE-Rank's ranking performance, with results presented in Table 3. As expected, the model encompassing all training stages and loss functions exhibited the highest performance across four datasets. Additionally, we make the following observations: firstly, the alignment stage markedly influences ranking performance, though a model with ranking capabilities can still be obtained without it. Secondly, adding text without the KL loss (row (d) vs. (c)) or merely incorporating the KL loss (row (e) vs. (c)) during training does not yield substantial improvements. Consequently, we infer that it is imperative for PE-Rank to comprehend the token-level interaction between query and passages, as well as to simulate the original text only using passage embeddings.

Different Embedding Models To verify whether our proposed framework can generalize to different embedding models, we choose a different embedding model for experiments. Specifically, we select BGE-base (Xiao et al., 2023), a BERT-based model that achieves the top tier position across the same parameter scale models on the MTEB benchmark (Muennighoff et al., 2022). We use BGE as the embedding model and the same complete train-

Model	*Ret.*	DL19	DL20	BEIR Avg.
BM25	BM25	50.58	47.96	43.80
PE-Rank$_{\text{Jina}}$		70.48	63.54	48.43
PE-Rank$_{\text{BGE}}$		67.28	63.52	47.91
Jina-Embeddings	Jina	65.94	63.89	41.46
PE-Rank$_{\text{Jina}}$		70.91	69.48	44.28
BGE-base	BGE	70.22	66.21	45.14
PE-Rank$_{\text{BGE}}$		72.93	67.80	46.00

Table 4: Using different embedding models to obtain passage embeddings as context compression.

Model	NDCG	w/s	#Proc.	Latency
RankMistral$_p$	71.96	20 / 10	19510.2	16.72
	60.26	40 / 20	17152.3	9.10
	51.54	100 / -	10561.9	4.09
PE-Rank	70.48	20 / 10	2942.4	3.68
	70.12	40 / 20	2187.7	3.05
	68.57	100 / -	1210.9	1.90

Table 5: The impact of different settings in the sliding window strategy on effectiveness and efficiency of reranking top 100 candidates retrieved by BM25.

ing process as Jina-Embeddings to obtain a new model. The results are shown in Table 4.

Firstly, using Jina-Embeddings and BGE as the encoder and leveraging their passage embeddings for reranking are both effective, reranking the candidates obtained from different retrieval models on different datasets can consistently bring improvement. This proves that the PE-Rank approach can be applied to different embedding models.

However, although BGE scores higher than Jina-embedding on MTEB, the performance of reranking BM25 retrieval results using BGE embeddings is consistently lower across three different datasets compared to using Jina embeddings. Due to the use of different training data and pooling methods in these two models, it is challenging to directly determine the cause of this discrepancy. Nonetheless, we have reason to believe that models excelling in general embedding benchmarks like MTEB may not necessarily perform well in this context. This issue is worth further investigation.

Impact of Sliding Window We investigate the effects of varying window sizes (w) and step sizes (s) in sliding window strategies, with results presented in Table 5. For RankMistral, ranking performance decreases sharply as window size increases. This is attributable to two factors: firstly, RankMistral struggles to manage long contexts containing rich information; secondly, it is trained on data with a window size of 20, which may prevent it from generating complete rankings with larger window sizes. In contrast, PE-Rank effectively addresses these issues. The compressed text maintains a shorter total length, and the compressed representation, i.e., passage embeddings, remains the key information of the original text. Additionally, the DC decoding method ensures accurate output of complete rankings. Consequently, PE-Rank's ranking performance remains relatively stable. More importantly, PE-Rank can reduce the number of sliding windows, thereby enhancing ranking efficiency.

5 Related Work

Large Language Models as Rerankers Recent advancements in large language models (LLMs) have shown their effectiveness in zero-shot reranking. There are three main paradigms for prompting LLMs: pointwise (Sachan et al., 2022; Liang et al., 2022), pairwise (Qin et al., 2023), and listwise (Sun et al., 2023; Pradeep et al., 2023a,b). Although pointwise is least effective and pairwise is inefficient, listwise achieves the best performance but is limited by context length and inference costs. PE-Rank aims to improve the efficiency of listwise approaches while maintaining their effectiveness.

Context Compression Aiming to reduce the input length of LLMs while retaining key information, there are some context compression approaches have been proposed, including heuristic modificationa (Jiang et al., 2023b) and dense memory slot compression (Chevalier et al., 2023; Ge et al., 2023; Mu et al., 2024). However, these are general methods and insufficient for ranking tasks. PE-Rank is designed for ranking and can be regarded as a variant of the soft prompts method, which can handle and compress multiple passages simultaneously for efficient listwise reranking.

6 Conclusion

In this paper, we propose a novel approach, PE-Rank, for efficient listwise passage reranking with large language models, leveraging passage embedding as the context compression, as well as effective inference and training methods. Experiment results demonstrate that PE-Rank offers significant efficiency advantages while achieving competitive reranking effectiveness.

7 Limitations

We acknowledge some potential limitations of this work. Firstly, for this method, we need to obtain passage embeddings and change the decoding space dynamically, resulting in a more complex architecture and additional memory allocation.

Secondly, this method is not plug-and-play, using different embedding models requires fine-tuning both MLP and LLM, rather than just MLP. We look forward to it being achieved by simply changing the MLP, thus making it easier to use.

Finally, due to resource limitations, the embedding models and LLMs we used are relatively small, and we have not conducted experiments on more models. It is still unclear how changing the model will affect this method. We leave the second and third points for future work.

References

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. 2024. xrag: Extreme context compression for retrieval-augmented generation with one token. *arXiv preprint arXiv:2405.13792*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846.

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*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yan Fang, Jingtao Zhan, Qingyao Ai, Jiaxin Mao, Weihang Su, Jia Chen, and Yiqun Liu. 2024. Scaling laws for dense retrieval. *arXiv preprint arXiv:2403.18684*.

Luyu Gao and Jamie Callan. 2021. Condenser: a pre-training architecture for dense retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 981–993.

Tao Ge, Jing Hu, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, et al. 2023. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

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. 2023a. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

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*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2023a. Fine-tuning llama for multi-stage text retrieval. *arXiv preprint arXiv:2310.08319*.

Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023b. Zero-shot listwise document reranking with a large language model. *arXiv preprint arXiv:2305.02156*.

Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested ranker. In *Proceedings of the*

29th annual international ACM SIGIR conference on Research and development in information retrieval, pages 437–444.

John Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M Rush. 2023. Text embeddings reveal (almost) as much as text. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12448–12460.

Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y Zhao, Yi Luan, Keith B Hall, Ming-Wei Chang, et al. 2021. Large dual encoders are generalizable retrievers. *arXiv preprint arXiv:2112.07899*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*.

Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424*.

OpenAI. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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*.

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

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

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*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

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. *arXiv preprint arXiv:2204.07496*.

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*, pages 14918–14937.

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*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Improving text embeddings with large language models. *arXiv preprint arXiv:2401.00368*.

Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighof. 2023. C-pack: Packaged resources to advance general chinese embedding. *arXiv preprint arXiv:2309.07597*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408*.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing dense retrieval model training with hard negatives. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1503–1512.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. Repbert: Contextualized text embeddings for first-stage retrieval. *arXiv preprint arXiv:2006.15498*.

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*.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023a. 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*, pages 2308–2313.

Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2023b. A setwise approach for effective and highly efficient zero-shot ranking with large language models. *arXiv preprint arXiv:2310.09497*.

A Additional Related Work

A.1 Multi-Stage Ranking

Multi-stage ranking, which can be traced back to work over a decade ago (Matveeva et al., 2006), aims to achieve both effective and efficient ranking. Current mainstream approaches generally adhere to a "*retrieval-then-rerank*" pipeline, first retrieve a set of candidates, followed by a more powerful reranker to enhance the ranking results (Nogueira et al., 2019; Ma et al., 2023a).

Dense retrieval models On the retriever side, dense retrieval models based on a bi-encoder design are prevalent. Based on the bi-encoder architecture, these models independently encode documents and queries into one dense embedding and use vector similarity to model the relevance (Karpukhin et al., 2020; Zhan et al., 2020). This design allows for the offline pre-encoding of the corpus, facilitating efficient retrieval during the search phase through approximate nearest neighbor search (ANNs) algorithms. Numerous techniques have been proposed to augment the efficacy of retrieval models, including mining hard negatives (Xiong et al., 2020; Zhan et al., 2021), pre-training for retrieval (Gao and Callan, 2021), large-scale contrastive training (Günther et al., 2023; Xiao et al., 2023), and scaling the model size (Ni et al., 2021; Fang et al., 2024; Wang et al., 2023). These methods improve the capacity of embeddings, enabling them to comprehensively capture the semantic information of the text.

Supervised neural rerankers On the reranker side, monoBERT (Nogueira et al., 2019) and monoT5 (Nogueira et al., 2020) demonstrated the effectiveness of employing pre-trained language models for reranking. RankT5 (Zhuang et al., 2023a) explored using ranking loss for training. These rerankers are usually trained on MS MARCO dataset.

Large Language Models as Rerankers Recently, large language models have demonstrated impressive effectiveness on many tasks. Many studies also attempt to utilize LLMs for zero-shot reranking. In general, there are three paradigms for prompting large language models: *pointwise*, *pairwise*, and *listwise*.

The pointwise approach evaluates the relevance score on one query-passage pair at a time, including *relevance generation* (Liang et al., 2022) and *query generation* (Sachan et al., 2022). The pairwise

approach prompts LLM with a pair of passages to a given query to indicate which is more relevant, using aggregation methods (Pradeep et al., 2021) or sorting algorithms (Qin et al., 2023; Zhuang et al., 2023b) to derive the final ranking.

The listwise approach aims to receive a query along with a list of candidates and directly generate a ranking list based on their relevance to the query (Ma et al., 2023b; Sun et al., 2023). Recently, some studies have attempted to distill smaller listwise reranking models from existing powerful rerankers like RankGPT (Pradeep et al., 2023a,b; Zhang et al., 2023).

Among these methods, the pointwise approach exhibits the poorest performance, the pairwise approach suffers from low efficiency, and only the listwise approach achieves optimal performance while maintaining a relatively reasonable efficiency level. However, it remains constrained by the context length and inference cost of LLMs. Our proposed method aims to enhance the efficiency of listwise approaches while preserving their effectiveness.

A.2 Context Compression

Context compression, which seeks to reduce the input length of LLMs while retaining the essential information from the original context, has recently garnered considerable attention. One approach is to heuristic modify the context to make it concise while retaining key information. LLMLingua (Jiang et al., 2023b) introduces a coarse-to-fine prompt compression method based on the perplexity score. RECOMP (Xu et al., 2023) proposes compressing documents into text summaries for RAG. Another direction is to compress the text into dense slots or soft prompts, such as AutoCompressor (Chevalier et al., 2023), ICAE (Ge et al., 2023), and Gist (Mu et al., 2024). However, these methods only compress a single prompt and are inadequate for ranking tasks. In contrast, our proposed method is specifically designed for ranking tasks and can be regarded as a variant of the soft prompts method.

Recently, a contemporary work, xRAG, proposed using embedding models to compress a document into a token for RAG, which is similar to our proposed method (Cheng et al., 2024). Compared to it, our proposed PE-Rank method has the following differences: firstly, we compress prompts for the ranking task which is more complex, and secondly, we compress multiple documents as input at once.

B Training Data

B.1 Dataset for Alignment

During the alignment stage, we employ segmented Wikipedia as the training dataset. The texts in the Wikipedia dataset, authored and reviewed by humans, are of higher quality and completeness. Additionally, its encyclopedic nature provides knowledge from diverse fields, rendering it reliable for training in the alignment stage. Specifically, we utilized the Wikipedia dump from Dec 2020, preprocessed by Izacard et al. (2023), which is totaling around 31.5 million texts. We sampled 2 million data pieces for training. The complete data format can be found in Appendix F.

B.2 Dataset for Learning-to-rank

In the learning-to-rank stage, we utilize the MS MARCO dataset (Bajaj et al., 2016). MS MARCO is a large-scale passage retrieval dataset that contains around 8.8 million passages and 800,000 queries, of which about 500,000 have manually annotated relevance labels.

We use Jina-embeddings-v2-base-en[1] as the retrieval model to retrieve the top 20 candidate passages for all queries in the training set, to construct the dataset. However, it only includes binary annotations (i.e., relevant or irrelevant) and cannot be directly used as training data for our training procedure. Therefore, following the approach of Zhang et al. (2023), we use an existing powerful supervised reranking model, i.e., MiniLM[2] trained on MS MARCO, as the annotation model to approximate the golden ranking. Following Pradeep et al. (2023a), we used a data augmentation strategy of randomly shuffling document order.

To facilitate training, we excluded samples with excessively long lengths, retaining only those with input lengths less than 2048. Consequently, our dataset for this stage comprises 232,419 samples and each sample contains 20 passages and the approximated golden ranking.

C Implementation Details

For the models we use, we select Mistral-7B-Instruct-v0.2 as the backbone.[3] For embedding

[1] https://huggingface.co/jinaai/jina-embeddings-v2-base-en

[2] https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2

[3] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

Hyperparameter	Alignment	LTR
optimizer	AdamW	AdamW
learning rate	1e-4	2e-5
lr scheduler type	cosine	cosine
warmup ratio	0.03	0.03
weight dacay	0	0
epochs	1	1
batch size per GPU	32	4
gradient accumulation	1	2
max sequence length	512	2048

Table 6: Hyperparameters for Training.

models, we use Jina-Embeddings and BGE-base[4] which are both encoder-based models with 137M parameters and 110M parameters, respectively. The selection of embedding models is based on the number of model parameters, their performance on MTEB, and community popularity. We didn't use top-tier models on MTEB because they are all decoder-based models that have a much larger number of parameters.

We implement all training codes based on the PyTorch framework. To optimize memory usage and accelerate training, we applied Deepspeed ZeRO stage 2 (Rasley et al., 2020) and BFloat16 mixed precision techniques. Additionally, Flash attention (Dao et al., 2022) was used to further improve training efficiency.

In Table 6, we present the hyperparameters for the alignment stage and learning-to-rank stage. All models were trained on 4 Nvidia H100 GPUs. The training for the alignment stage required approximately 7 hours, while the learning-to-rank stage also took 7 hours. It is important to note that the hyperparameters were determined based on empirical observations, as comprehensive hyperparameter tuning was beyond the scope of this study due to resource constraints.

D Selection of Baselines

We provide a detailed introduction to the selection of baselines here.

Supervised Neural Rerankers First, we select two typical supervised models, including:

- **monoBERT** (Nogueira and Cho, 2019), a cross-encoder based on BERT-Large (Devlin et al., 2018), which uses the concatenation of the query and the passage as input and maps the embedding of [CLS] token to a score.
- **monoT5** (Nogueira et al., 2020), which is a sequence-to-sequence reranking model based on T5-3B (Raffel et al., 2020), using the probability of the output token "true" as the relevance score.

These two models are both trained on the MS MARCO dataset using a large number of human annotation labels.

LLM-based Rerankers Additionally, we use one unsupervised LLM-based methods as baselines:

- **RankGPT** (Sun et al., 2023), a state-of-the-art listwise method that uses a sliding window strategy for listwise ranking based on GPT.

We also add listwise reranking models that are based on smaller LLMs (such as an LLM with 7B parameters) and are distilled from existing rerankers. In particular, we select:

- **RankVicuna** (Pradeep et al., 2023a), which is a listwise model based on Vicuna-7B, using $\text{RankGPT}_{3.5}$ as the teacher model.
- **RankZephyr** (Pradeep et al., 2023b), which is a listwise model based on a more powerful backbone Zephyr-7B, using both $\text{RankGPT}_{3.5}$ and RankGPT_4 as the teacher model thus achieve a strong ranking performance.

Besides, we also use a ranking model trained by ourselves. The training process is similar to RankVicuna but uses the data mentioned in the previous section. This decision is motivated by two reasons. Firstly, the choice of the base model can significantly influence the performance of the ranking model. Secondly, the selection of different teacher models can have a substantial impact. Consequently, to ensure a more equitable comparison, we retrained a ranking model based on Mistral-7B as the baseline, denoted as $\textbf{RankMistral}_*$.

We replace $*$ with different forms of text input, including:

- $\textbf{RankMistral}_p$, which use original passage as the input.
- $\textbf{RankMistral}_s$, which use the summary to relace the passage. The summary is generated by Mistral-7B-Instruct-v0.2.

[4] https://huggingface.co/BAAI/bge-base-en-v1.5

Model	*Ret.*	Covid	NFCorpus	Touché	DBPedia	SciFact	Signal	News	Robust	Avg.
BM25	-	59.47	33.75	44.22	31.80	67.89	33.05	39.52	40.70	43.80
monoBERT	BM25	70.01	36.88	31.75	41.87	71.36	31.44	44.62	49.35	47.16
monoT5	BM25	80.71	38.97	32.41	44.45	76.57	32.55	48.49	56.71	51.36
$\text{RankGPT}_{3.5}$	BM25	76.67	35.62	36.18	44.47	70.43	32.12	48.85	50.62	49.37
RankGPT_{4}	BM25	85.51	38.47	38.57	47.12	74.95	34.40	52.89	57.55	53.68
RankMistral_{p}	BM25	78.00	33.10	27.46	37.71	66.22	30.04	37.10	39.54	43.65
$\text{PE-Rank}_{\text{Jina}}$	BM25	77.72	36.39	33.06	40.05	69.38	33.74	49.70	47.40	48.43
$\text{PE-Rank}_{\text{BGE}}$	BM25	77.21	36.24	35.68	38.91	69.29	32.86	47.94	45.12	47.91
Jina-Embeddings	-	68.94	31.43	28.68	33.32	65.53	25.76	39.80	38.23	41.46
RankMistral_{p}	Jina	80.19	29.74	29.16	40.25	63.85	28.17	35.80	35.69	42.86
$\text{PE-Rank}_{\text{Jina}}$	Jina	77.49	30.92	30.00	36.26	64.48	26.54	44.78	43.73	44.28
BGE	-	75.19	36.58	23.64	37.21	74.41	28.18	41.93	43.96	45.14
RankMistral_{p}	BGE	82.75	34.99	27.80	43.04	72,72	29.02	39.06	40.51	42.45
$\text{PE-Rank}_{\text{BGE}}$	BGE	80.56	36.94	24.26	39.84	71.88	26.20	44.18	44.13	46.00

Table 7: Full results on BEIR benchmark. For all datasets, NDCG@10 is used as the metric.

Model	n	NDCG	# Proc.	Latency (s)
RankMistral_{p}	20	60.64	2190.4	1.88
RankMistral_{s}		59.52	1446.8	1.81 (×.97)
RankMistral_{t}		42.28	422.8	1.85 (×.99)
PE-Rank		56.48	327.7	0.42 (×.22)
RankMistral_{p}	100	68.39	19787.0	16.31
RankMistral_{s}		66.14	13514.3	15.74 (×.97)
RankMistral_{t}		38.15	3874.9	15.29 (×.94)
PE-Rank		63.54	2949.0	3.66 (×.22)

Table 8: Efficiency analysis or reranking top n candidates retrieved by BM25 on TREC DL20.

Model	Order	TREC DL19	TREC DL20
BM25	-	50.58	47.96
RankMistral_{p}	Origin	71.73	68.07
	Random	71.04	67.91
	Inverse	70.92	68.85
PE-Rank	Origin	70.48	63.54
	Random	66.74	56.14
	Inverse	57.15	49.85

Table 9: Sensitivity to the initial ranking.

- **RankMistral**$_t$, which use the title obtained from the origincal datasets.

These baselines help us evaluate the effectiveness and efficiency of different compression methods under a consistent setting.

We didn't include other context compression methods as baselines for efficiency analysis because they are unsuitable for ranking tasks.

E More Results

We give more evaluation results and analysis here.

E.1 Full Results on BEIR

Table 7 shows the full results on BEIR benchmark.

E.2 Efficiency Analysis on TREC DL20

Table 8 shows the analysis results on TREC DL20.

E.3 Sensitivity to the Initial Ranking

We analyze different orderings of the candidates that are retrieved by BM25, including the original BM25 ranking order, inverted BM25 ranking order, and random shuffled order. The results are shown in Table 9. We can see that compared to using passage as input, using embeddings as the compressed input may be more sensitive to the initial order. This may be one of the limitations of this method.

F Prompts

Alignment Stage Training For alignment stage, we use diverse instruction data, shown in Table 10.

Learning-to-rank Stage Training For learning-to-rank stage, as discussed in Section 2.3, we used two different types of training data. The full data formats are listed in Table 11 and Table 12.

Training RankMistral The prompt used for training RankMistral is listed in Table 13.

Generating Summaries The prompt for generating summaries for RankMistral_s is in Table 14.

Prompts for Evaluation For RankMistral_*, we use the same prompt as training shown in Table 13. For PE-Rank, we use the prompt shown in Table 11.

User:

- Given the passage: {{embedding}}, reconstruct the original text.
- Passage: {{embedding}} means the same as
- Passage: {{embedding}} Can you say the above text again?
- {{embedding}} Please provide a reconstruction of the preceding passage.
- Passage: {{embedding}} is about what?
- {{embedding}} Could you give me a different version of the passage above?
- Passage: {{embedding}} Please offer a restatement of the provided passage.
- Passage: {{embedding}}, which means:

Assistant:
{{text}}

Table 10: Prompts used for alignment stage training, where {{embedding}} and {{text}} are placeholders for transformed embeddings $\mathbf{E}_M(\boldsymbol{e}_t)$ and the original text t.

User:

I will provide you with {{n}} passages, each with a special token representing the passage enclosed in [].

Rank the passages based on their relevance to the search query: {{query}}.

Passage 1: [{{embedding}}]

...

Passage {{n}}: [{{embedding}}]

Search Query: {{query}}

Rank the {{n}} passages above based on their relevance to the search query in descending order. Only output the {{n}} unique special token in the ranking.

Table 11: Data format used for learning-to-rank stage training.

User:
I will provide you with {{n}} passages, each with a special token representing the passage enclosed in [], followed by the original text.

Rank the passages based on their relevance to the search query: {{query}}.

Passage 1: [{{embedding}}] {{content}}

...

Passage {{n}}: [{{embedding}}] {{content}}

Search Query: {{query}}

Rank the {{n}} passages above based on their relevance to the search query in descending order. Only output the {{n}} unique special token in the ranking.

Table 12: Data format used for learning-to-rank stage training.

User:

I will provide you with {{n}} passages. Rank the passages based on their relevance to the search query: {{query}}.
Passage 1: {{content}}

...

Passage {{n}}: [{{embedding}}] {{content}}

Search Query: {{query}}

Rank the {{n}} passages above based on their relevance to the search query in descending order. The output format should be [] > [] > ..., e.g., [4] > [2] > ..., Only respond with the ranking results with {{n}} unique numbers, do not say anything else or explain.

Table 13: Data format used for training RankMistral.

User:
Summarize the following passage, only output the summary, do not include anything else.
Passage: {{content}}

Table 14: Prompts used for generating summary using Mistral-7B.