ListConRanker: A Contrastive Text Reranker with Listwise Encoding

Anonymous ACL submission

Abstract

Reranker models aim to re-rank the passages based on the semantics similarity between the given query and passages, which have recently received more attention due to the wide application of the Retrieval-Augmented Generation. Most previous methods apply pointwise encoding, meaning that it can only encode the context of the query for each passage input into the model. However, for the reranker model, given a query, the comparison results between passages are even more important, which is called listwise encoding. Besides, previous models are trained using the cross-entropy loss function, which leads to issues of unsmooth gradient changes during training and low training efficiency. To address these issues, we propose a novel **List**wise-encoded **Con**trastive text re**Ranker** (**ListConRanker**). It can help the passage to be compared with other passages during the encoding process, and enhance the contrastive information between positive examples and between positive and negative examples. At the same time, we use the circle loss to train the model to increase the flexibility of gradients and solve the problem of training efficiency. Experimental results show that ListConRanker achieves state-of-the-art performance on the reranking benchmark of Chinese Massive Text Embedding Benchmark, including the cMedQA1.0, cMedQA2.0, MMarcoReranking, and T2Reranking datasets.[1]

1 Introduction

Given a query and a few passages that are semantically related or partially related to the query, the goal of the ranking task is to sort these passages based on their degree of semantic similarity. The reranker model is one of the most important parts in the Information Retrieval (IR) systems (Guo et al., 2020; Pang et al., 2020; Padaki et al., 2020). Recently, due to the trend of Retrieval-Augmented Generation (RAG) and limited context length of Large Language Models (LLMs), reranking models have been widely studied and rapidly developed (Lewis et al., 2020; Zhao et al., 2024; Glass et al., 2022), which can filter out noisy passages retrieved by embedding models.

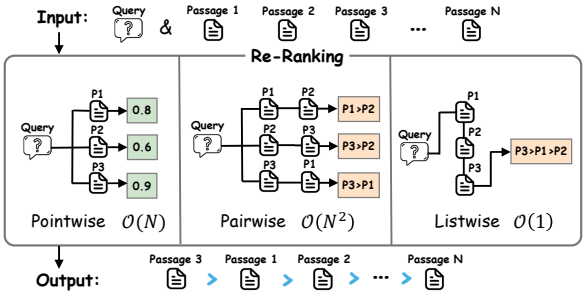

Figure 1: Pointwise rerankers receive the query and a passage as input and output the similarity score of them. Pairwise rerankers receive the query and two passages as input and output which passage is more similar to the query. Listwise rerankers receive the query and all passages as input and directly output the final ranking results. The time complexity is calculated under ideal conditions, without taking into account the models used.

As shown in Figure 1, there are three main types of reranking models, which are pointwise (Nogueira et al., 2020; Liang et al., 2023; Sachan et al., 2022), pairwise (Qin et al., 2024), and listwise (Sun et al., 2023; Ma et al., 2023). The main difference between them is that a pointwise reranker accepts a query and one passage as input at a time, while a pairwise reranker takes two passages, and a listwise reranker takes multiple passages at a time.

Most previous rerankers based on BERT-like Cross-Encoder are pointwise (Lu et al., 2022), because of the limited context length and the need to align with the two-sentence input format used during pre-training (Devlin et al., 2019; Zhuang et al., 2021). However, pointwise rerankers can only obtain the context of the query and a passage,

[1]The weights and codes of ListConRanker will be released after acceptance.

when models encode the feature of passage. The most important aspects are the relative ordering of passages and the comparison of similarities between passages. This results in the suboptimal performance of pointwise rerankers. Pairwise and listwise encoding reranker are two solutions to this problem. Due to the high computational complexity and unstable comparison results (e.g., passage 1 > 2 and passage 2 > 3, but passage 3 > 1) of pairwise methods, their practical application value is low. In contrast, listwise methods represent a more realistic and valuable direction for exploration.

Recently, LLMs have demonstrated excellent performance on natural language tasks. Some work has attempted to use LLMs as listwise rerankers (Pradeep et al., 2023; Liu et al., 2024). They allow the LLM to directly output the predefined sequence numbers of passages as the final ranking result. However, it will still exceed the context length limit when there are hundreds of passages, leading to the failure of ranking (Sun et al., 2023). Besides, LLMs can not output exact similarity scores between the query and passages. Moreover, different input orders may also lead to instability in the results.

Based on the problems of the pointwise reranker and the listwise reranker based on LLMs, we explore the listwise reranker based on BERT-like embedding models, which is an area that has not been fully explored before. We propose a **List**wise-encoded **Con**trastive text re**Ranker** (**ListConRanker**). It first inputs query and passages into the embedding model. After getting the original features of the query and each passage, we combine these features into an input sequence, which is then fed into the proposed ListTransformer. The ListTransformer can facilitate global contrastive information learning between passage features, including the clustering of similar passages, the clustering between dissimilar passages, and the distinction between similar and dissimilar passages. Besides, we propose ListAttention to help ListTransformer maintain the features of the query while learning global comparative information.

At the same time, most of the previous pointwise rerankers apply cross-entropy (CE) loss function during training. However, for the ranking task, there is usually more than one similar (positive) passage. If the CE loss function is used, for multiple passages of the same query, models can only sample one positive passage per training step. This leads to low training efficiency for positive samples. Besides, it hinders the learning of contrastive information between positive samples.

To this end, we propose to apply Circle Loss (Sun et al., 2020). It can improve the data efficiency by taking multiple positive and negative passages as input at the same time. Besides, its self-adaptive weights can smooth out gradient changes during the training process and accelerate training. Specifically, it can reduce the gradient when close to the optimal space while increasing the gradient when far from it. This helps the model find the optimal space more quickly.

The main contributions of our work can be summarized as follows:

- We propose a novel BERT-based reranker model with listwise encoding named ListConRanker, which has not been fully explored before. It contains ListAttention and ListTransformer, which can utilize the global contrastive information to learn representations.
- We propose to use the Circle Loss as the loss function. Compared with CE loss and ranking loss, it can solve the problems of low data efficiency and unsmooth gradient change.
- Experimental results on the reranking benchmark of Chinese Massive Text Embedding Benchmark (C-MTEB) demonstrate that ListConRanker is state-of-the-art. The ablation study shows the effectiveness of ListTransformer and Circle Loss.

2 Related Work

There are three main encoding types of rerankers: pointwise encoding, pairwise encoding, and listwise encoding. As shown in Figure 1, the main differences between them are the number of passages input at a time and the output format. Due to the high computational complexity of pairwise rerankers, which is $O(N^2)$ for ranking N passages, there has been less related work in the past. In the following sections, we will mainly introduce these three rerankers.

2.1 Pointwise Rerankers

Pointwise rerankers receive the query and a passage as input, and output the similarity between them. Dai and Callan (2019) and Gao et al. (2021) were the first to propose using the BERT model in ranking tasks and referred to this type of model

as the cross-encoder architecture. This allows for interaction between the query and the passage. However, they used the binary cross-entropy loss function during training, which prevented interaction between query-passage pairs. Based on this, Nogueira and Cho (2019) proposed using contrastive loss from a new perspective. Recently, due to the huge success of generative language models, Nogueira et al. (2020) proposed to input the query and passage into the model directly and then let the model generate a "true" or "false" token to indicate whether they are similar. Furthermore, Sachan et al. (2022) proposed to use LLMs to compute the probability of the input query conditioned on the passage as the similarity between them.

However, these pointwise rerankers can only learn the context of one passage, which results in suboptimal performance. Listwise rerankers can provide global contrastive information, making it superior in architecture.

2.2 Pairwise and Listwise Rerankers

Pairwise Rerankers take a query and two passages as input and output which of the two passages is more similar to the query. Qin et al. (2024) proposed using an LLM as a judge. After obtaining results for all passage pairs, a traditional ranking algorithm is used to produce the final passage ranking.

Listwise rerankers receive the query and all corresponding passages as input. It outputs the ranking of passages directly. Most previous methods are based on LLMs. For example, some methods first proposed to let LLMs generate the reordered list (Ma et al., 2023; Pradeep et al., 2023; Zhang et al., 2023). To address the issue of exceeding the context length of LLMs in some extreme cases, Sun et al. (2023) proposed a listwise method based on sliding windows. Furthermore, Liu et al. (2024) encoded the passage into an embedding before inputting it into the LLM. This helps to address the lack of global contrastive information in the methods based on sliding windows.

Due to the limited context length of LLMs, the rerankers based on LLMs can not provide complete global contrastive information, which is the most important advantage of listwise rerankers.

3 Method

In this section, we mainly describe ListConRanker, which applies listwise encoding to learn global contrastive information. It uses the Circle Loss (Sun et al., 2020) to improve the data efficiency and smooth out gradient changes. The structure of ListConRanker is shown in Figure 2.

3.1 Original Features

Given a query q and N passages $P = \{p_1, p_2, ..., p_N\}$, we first input them to an embedding model to obtain their features $H = \{h_q, h_1, h_2, ..., h_N\}$.

$$h_q = \text{Embedding}(q) \quad (1)$$

$$h_i = \text{Embedding}(p_i) \quad (2)$$

where Embedding is a BERT-based embedding model, h_q is the feature of query, and h_i is the feature of passage i.

We get the original features by using an embedding to encode the context of the query and passage. However, these features do not contain any information of other passages, which means these passages are unable to be compared with other passages to optimize the ranking result.

3.2 Listwise Encoding by ListTransformer and ListAttention

After obtaining the features of query and passages, we concatenate these features into a sequence and input them into the ListTransformer which is similar to a Transformer Encoder (Vaswani et al., 2017) without position embedding except for the self-attention module. In addition, we apply learnable embeddings to help ListTransformer distinguish between query feature and passage features.

$$Z^{(0)} = \{h_q + e_q, h_1 + e_p, ..., h_n + e_p\} \quad (3)$$

$$Q^{(l)} = Z^{(l-1)} W_Q^{(l)} \quad (4)$$

$$K^{(l)} = Z^{(l-1)} W_K^{(l)} \quad (5)$$

$$V^{(l)} = Z^{(l-1)} W_V^{(l)} \quad (6)$$

$$Z^{(l)} = \text{ListAttention}(Q^{(l)}, K^{(l)}, V^{(l)}) \quad (7)$$

where e_q and e_p are learnable embeddings, $W_Q^{(l)}$, $W_K^{(l)}$, and $W_V^{(l)}$ are learnable parameters, and l is the l-th layer of ListTransformer.

The bidirectional self-attention in ListTransformer will make the query feature obscured by unrelated passages when there are a large number of unrelated passages. When the number of unrelated passages is small, the query can ignore the

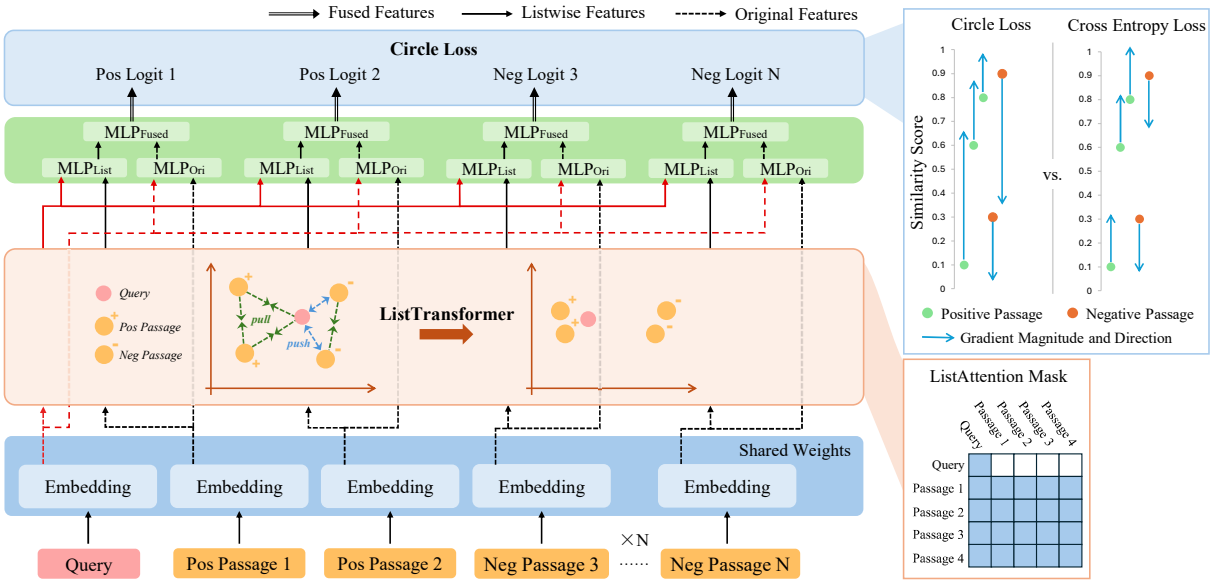

Figure 2: The overall architecture of the ListConRanker.

unrelated passages by attention weights. However, when the number is large, the noisy information dominates self-attention.

To this end, we propose the ListAttention in the ListTransformer. The attention mask details of ListAttention are shown in Figure 2. Specifically, we make the query can only get its own attention to preserve its original features. As for passages, we use the bidirectional self-attention. Passages can recognize the query features by query embedding and learn pair features between query and passage. Besides, the attention among passages can facilitate the learning of global contrastive information between passages. For example, it can reduce the distance between positive examples and between negative examples while increasing the distance between positive and negative examples. This global contrastive information reflects the concept of sequencing in ranking tasks.

After obtaining the features with global contrastive information (i.e., listwise features), we combine the listwise features with original features and use MLPs to determine the final similarity between the query and passage.

$$s_i^{origin} = \text{MLP}_{\text{Ori}}(h_q, h_i) \tag{8}$$

$$s_i^{list} = \text{MLP}_{\text{List}}(z_q^{(l)}, z_i^{(l)}) \tag{9}$$

$$s_i^{final} = \sigma(\text{MLP}_{\text{Fused}}(s_i^{origin}, s_i^{list})) \tag{10}$$

where $S^{final} = \{s_1^{final}, s_2^{final}, ..., s_n^{final}\}$ is the similarity between the query and passage p_i, MLP_{Ori}, MLP_{List}, and $\text{MLP}_{\text{Fused}}$ are all MLPs, σ is the sigmoid activation function.

Finally, we can sort all passages by their similarity to the query S^{final} in descending order to obtain the final ranking result.

3.3 Circle Loss

Most of the previous rerankers apply cross-entropy loss function to train the model. However, cross-entropy loss function has several disadvantages. First, its data efficiency is low. It can only sample one similar (positive) passage from all the passages per training step. This also prevents global contrast information learning between positive examples. Second, it does not have the feature of adaptively adjusting loss weights. Specifically, when the predicted values are close to or far from having the optimal space, the cross-entropy loss function can only implicitly control the size of parameter updates through the magnitude of the gradient. At the same time, the ordering is a relative result in ranking tasks. It only requires that positive samples be ranked higher than negative samples. The cross-entropy loss function requires the predictions for positive cases to be 1 and for negative cases to be 0. This is too strict for ranking tasks.

Therefore, we propose using Circle Loss (Sun et al., 2020) as the loss function for ListConRanker. It can customize the optimal prediction values for positive and negative cases. At the same time, it can adaptively adjust the gradient weights for each sample based on the customized optimal prediction values, which helps the model achieve smoother

gradient changes. Ultimately, this makes it easier for the model to find the optimal space. The calculation method for Circle Loss is as follows:

$$L = \log(1 + r_{neg} r_{pos}) \tag{11}$$

$$r_{neg} = \sum_{j=1}^{J} \exp(\gamma \alpha_{neg}^{j} (s_{neg}^{j} - \Delta_{neg})) \tag{12}$$

$$r_{pos} = \sum_{i=1}^{I} \exp(-\gamma \alpha_{pos}^{i} (s_{pos}^{i} - \Delta_{pos})) \tag{13}$$

$$\Delta_{neg} = m,\ \Delta_{pos} = 1 - m \tag{14}$$

$$O_{neg} = -m,\ O_{pos} = 1 + m \tag{15}$$

$$\alpha_{neg}^{j} = \max(0, s_{neg}^{j} - O_{neg}) \tag{16}$$

$$\alpha_{pos}^{i} = \max(0, O_{pos} - s_{pos}^{i}) \tag{17}$$

where m and γ are hyper-parameters, I is the number of positive samples in all passages, J is the number of negative samples in all passages.

3.4 Iterative Inference

Due to the limited memory of GPUs, we input about 20 passages at a time for each query during training. However, during actual use, there may be situations where far more than 20 passages are input at the same time. For example, each query corresponds to approximately 1,000 passages that need to be ranked in the MMarcoReranking (Bonifacio et al., 2021) test set. The differences between the training and inference processes can cause self-attention weights to become dispersed in ListTransformer, which prevents the passages from learning effective global contrastive information. This has been fully demonstrated in the context length extension problem of large language models (Chen et al., 2023; Press et al., 2022). Finally, it leads to a drop in performance.

However, we find that ListConRanker still maintains a basic ranking capability when a large number of passages are input simultaneously through case studies. Specifically, for some clearly dissimilar passages, ListConRanker can still rank them after partially similar samples (i.e., hard negative samples) or similar samples (positive samples). When a large number of passages are input, ListConRanker shows insufficient discrimination ability between partially similar samples or similar samples. On the contrary, when only dozens of passages are input, ListConRanker can easily distinguish between partially similar passages and similar samples through global contrastive information.

Algorithm 1 Iterative Inference

Input: Query q, Passage $P = \{p_1, p_2, ..., p_N\}$, Termination condition of the iteration θ, Passage reduction rate per iteration β
Output: Ranked result $R = \{r_1, r_2, ..., r_N\}$, where r_i is the order of p_i

```
while |P| > θ do
    nums ← ceil(|P| × β)
    scores ← ListConRanker(q, P)
    orders ← argsort(scores) // Descending
    for each i ∈ [1, nums] do
        last_one ← orders[−i]
        r_last_one ← |P|
        P.remove(p_last_one)
    end for
end while
scores ← ListConRanker(q, P)
orders ← argsort(scores) // Descending
for each i ∈ [1, |P|] do
    last_one ← orders[−i]
    r_last_one ← |P|
    P.remove(p_last_one)
end for
return R
```

Based on the above findings, we propose a novel iterative inference method for ListConRanker. The specific inference process is shown in Algorithm 1 and Figure 3. For each inference step i, we input all N_i passages and then determine the ranking positions for the bottom $(\beta \times N_i)$ passages in the final results. The remaining $N_{i+1} = (1 - \beta) \times N_i$ passages are re-input into the ListConRanker. The iteration continues until the number of remaining passages is less than θ. The final positions for all passages are obtained.

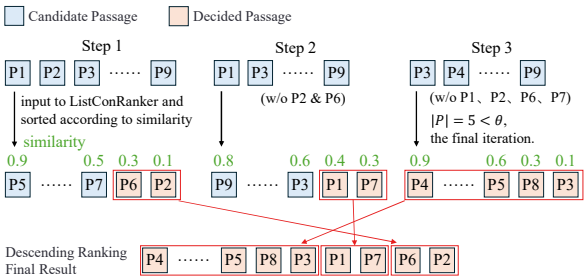

Figure 3: The process of iterative inference. For simplicity, we do not show the query in the figure. The query will be input during each step of the inference.

Model	LLM?	cMedQA1.0	cMedQA2.0	MMarcoReranking	T2Reranking	Avg.
360Zhinao-search	✘	87.00	88.48	32.41	67.80	68.92
Baichuan-text-embedding	✘	88.06	88.46	34.30	67.85	69.67
Yinka	✘	89.26	90.05	32.74	67.05	69.78
piccolo-large-zh-v2	✘	89.31	90.14	33.39	67.15	70.00
360Zhinao-1.8B-Reranking	✔	86.75	87.92	37.29	68.55	70.13
ternary-weight-embedding	✘	88.74	88.38	35.04	68.39	70.14
LdIR-Qwen2-reranker-1.5B	✔	86.50	87.11	39.35	68.84	70.45
zpoint-large-embedding-zh	✘	91.11	90.07	38.87	**69.29**	72.34
xiaobu-embedding-v2	✘	90.96	**90.41**	39.91	69.03	72.58
Conan-embedding-v1	✘	**91.39**	89.72	41.58	68.36	72.76
ListConRanker	✘	90.55	89.38	**43.88**	69.17	**73.25**
- w/o Iterative Inference		90.19	89.93	37.52	69.17	71.70

Table 1: The results comparison with baselines on the reranking benchmark of C-MTEB. We select the top 10 open-source models or models providing API inference as of December 10, 2024, based on the average metrics across cMedQA1.0, cMedQA2.0, MMarcoReranking, and T2Reranking as the baselines. The evaluation metric for all datasets is mAP. The best performance is in **bold**.

4 Experiments

4.1 Datasets and Evaluation Metrics

We evaluate ListConRanker on the reranker section of Chinese Massive Text Embedding Benchmark (C-MTEB)[2] (Xiao et al., 2024), including the test set of cMedQA1.0 (Zhang et al., 2017), test set of cMedQA2.0 (Zhang et al., 2018), development set of MMarcoReranking, and development set of T2Reranking (Xie et al., 2023). We report the mean Average Precision (mAP) on each dataset to show the effectiveness of reranker models.

4.2 Implementation Details

We initialized the embedding model using the Conan-embedding-v1[3] model (Li et al., 2024). Since ListTransformer and MLPs are trained from scratch, we trained the model in two stages. Firstly, we freeze the embedding model and only train the ListTransformer and MLPs for 4 epochs using the training sets of cMedQA1.0, cMedQA2.0, MMarcoReranking, T2Reranking, huatuo(Li et al., 2023), MARC(Keung et al., 2020), XL-sum(Hasan et al., 2021), and CSL(Li et al., 2022) with a batch size of 1024. Secondly, we do not freeze any parameter and use the training sets of cMedQA1.0, cMedQA2.0, and T2Reranking to train for 2 epochs with a batch size of 256. Besides, we respectively set the hyperparameters l, γ, θ, and β to 2, 10, 20, 0.2. In the first stage of training, we set m to -0.2. In the second stage of training, we set m to 0.1. The experiments are run on 8 NVIDIA A800 40GB GPUs with about 260 A800 GPU hours.

[2] https://huggingface.co/spaces/mteb/leaderboard

[3] https://huggingface.co/TencentBAC/Conan-embedding-v1

4.3 Overall Results

Table 1 shows the results of cMedQA1.0, cMedQA2.0, MMarcoReranking, and T2Reranking. Compared to other models, ListConRanker demonstrated a significant advantage, particularly on the MMarcoReranking and T2Reranking datasets. Although some previous methods are based on LLMs with larger parameter sizes (i.e., 360Zhinao-1.8B-Reranking and LdIR-Qwen2-reranker-1.5B), they use pointwise encoding, which leads to suboptimal performance. It is worth noting that we do not use the MMarcoReranking dataset in the second stage of training. The significant improvement of ListConRanker on the MMarcoReranking can be attributed to the listwise encoding introduced by the ListTransformer. Additionally, the MMarcoReranking is unique compared to the other three datasets. Each query in the MMarcoReranking corresponds to 1,000 passages. This large number of passages allows for extensive global contrastive information and interaction within the ListTransformer, resulting in a performance boost.

4.4 Ablation Study

4.4.1 The Effect of ListAttention

To verify the effectiveness of ListAttention, we conducted experiments with different attention masks. The results, as shown in Table 2, indicate that bidirectional self-attention performs worse than ListAttention. The only difference between them lies in whether the query needs to compute attention for

Model	cMedQA1.0	cMedQA2.0	MMarcoReranking	T2Reranking	Avg.
ListAttention	90.55	89.38	**43.88**	69.17	**73.25**
Bidirectional Self-Attention	**90.79**	**89.59**	42.12	**69.20**	72.93
PassageAttention	89.73	89.46	41.08	69.04	72.33

Table 2: The results of ablation study using different attentions on the reranker section of C-MTEB. All results are based on iterative inference. The best performance is in **bold**.

Model	cMedQA1.0	cMedQA2.0	MMarcoReranking	T2Reranking	Avg.
Fusing Original and Listwise Features	**90.55**	**89.38**	**43.88**	**69.17**	**73.25**
Only Listwise Features	90.36	**89.38**	41.48	68.91	72.53
Only Original Features	89.41	89.13	38.66	68.72	71.48
Embedding Model Continue Training	89.02	88.54	38.12	69.11	71.20

Table 3: The results of ablation study using different features on the reranker section of C-MTEB. The results of Fusing Original and Listwise Features and Only Listwise Features are based on iterative inference. Iterative inference will not change the results of Only Original Feature and Embedding Continue Training, as the features of each passage are not influenced by other passages. Thus, there is no need to use iterative inference on them. The best performance is in **bold**.

the passages and whether the query features are influenced by the passages. This decline in performance highlights the importance of maintaining the semantic features of the query.

Additionally, we explored a PassageAttention (see details in Appendix G). Specifically, in PassageAttention, the query has bidirectional self-attention with all passages, but each passage only computes attention between itself and the query. Passages rely on the query as an anchor to learn global comparative information. As a result, query features are influenced by all passages, regardless of their similarity to the query. This causes the query to lose its own semantic meaning. Meanwhile, it also reduces the efficiency of passages learning global contrastive information. Notably, PassageAttention resulted in the lowest performance, further highlighting the importance of maintaining the semantic features of query.

4.4.2 The Effect of Fusing Original and Listwise Features

To verify the effect of fusing listwise features with original features, we conducted experiments where we used only listwise features and only original features separately. The results, shown in Table 3, demonstrate that using only listwise features results in only a slight performance drop compared to ListConRanker. However, when the model uses only original features, there is a significant drop in performance, especially on the cMedQA1.0 and MMarcoReranking datasets. This is because the original features lack global comparative information, as they are pointwise features. This underscores that listwise features play a dominant role in ListConRanker, as well as the importance of global comparative information.

Additionally, to rule out the impact of data and training strategies, we further train the embedding model using the second-stage training strategy. We do not use the first-stage strategy, because the embedding model is frozen in the first stage. Since similarity calculation uses inner products between vectors, there is no feature interaction between the query and passage, meaning it is neither a pointwise nor a listwise model. As observed, its performance is the lowest, further emphasizing the importance of query-passage interaction in the reranker.

4.4.3 The Effect of Circle Loss

To verify the effectiveness of Circle Loss, we explored the experimental results using different loss functions, as shown in Table 4. Firstly, when using the cross-entropy loss, the model achieved the worst performance due to inefficient data sampling. Secondly, when the loss function was switched to Triplet Loss (Balntas et al., 2016) and CoSENT Loss[4], which are designed specifically for ranking tasks, the models are able to sample all positive and negative samples of query in each training step. This not only improved the data sampling efficiency but also enabled the ListTransformer to bring positive samples closer together, which obtains more comprehensive global contrastive information. Finally, when the model used Circle Loss, compared

[4] https://spaces.ac.cn/archives/8847

Model	cMedQA1.0	cMedQA2.0	MMarcoReranking	T2Reranking	Avg.
Circle Loss	**90.55**	89.38	**43.88**	**69.17**	**73.25**
CoSENT Loss	90.48	**89.79**	42.64	68.99	72.98
Triplet Loss	90.13	88.71	42.04	68.85	72.43
Cross-Entropy Loss	89.05	88.95	35.39	66.65	70.01

Table 4: The results of ablation study using different loss functions on the reranker section of C-MTEB. All results are based on iterative inference. The best performance is in **bold**.

to Triplet Loss and CoSENT Loss, the model has smoother gradient changes during training. Specifically, Circle Loss increases the gradient when the model is far from the optimal space and reduces the gradient as it approaches the optimal space. This allows the model to find the optimal space more easily, leading to the best performance.

4.4.4 The Effect of Iterative Inference

To explore the effect of iterative inference, we compared the results of ListConRanker using iterative inference with those using non-iterative inference. The results are shown in Table 1. In the non-iterative inference approach, we input all passages into the ListConRanker at once. And we directly sort the passages based on their output scores to obtain all the results. It can be observed that iterative inference improves performance on both the cMedQA1.0 and MMarcoReranking datasets, especially on MMarcoReranking. This is due to the fact that each query in MMarcoReranking corresponds to a large number of candidate passages, as mentioned above. In contrast, iterative inference has a slight negative effect on cMedQA2.0. This might be because all the negative passages in the cMedQA2.0 are very similar to the query, causing the positive passages to be ranked lower in the early iterations. Further, this prevents them from being input into the final iteration.

4.4.5 The Effect of Scaling Up ListTransformer

To explore whether ListConRanker has a scaling-up ability similar to LLMs (Kaplan et al., 2020), we conducted experiments with different numbers of ListTransformer layers. And the results are shown in Figure 4. However, as the number of layers increased, we do not observe a significant improvement in the performance of ListConRanker. In most cases, the number of passages corresponding to a query is small. Therefore, a smaller number of ListTransformer layers is already sufficient for effective comparison among passages and learning global comparative information. Moreover, this

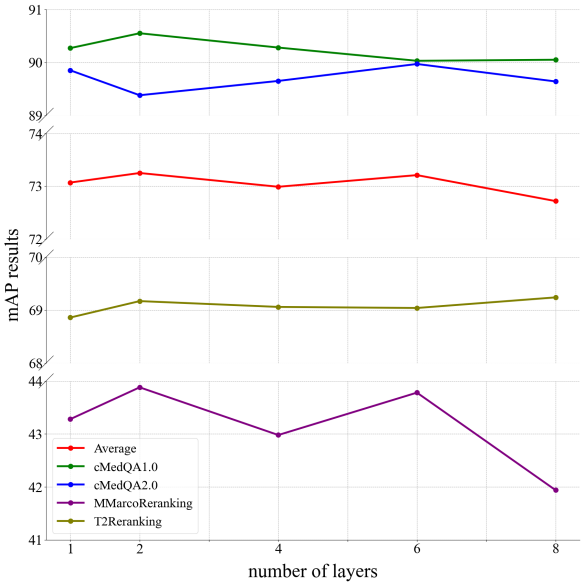

Figure 4: The influence of different layers of ListTransformer.

also demonstrates the robustness of ListConRanker to the layer of ListTransformer. ListConRanker outperformed the state-of-the-art model across all layer settings except for the 8-layer configuration, which might be due to the lack of an increase in the training data size alongside the training parameters. This caused the model to be undertrained.

5 Conclusion

In this paper, we propose a novel reranker named ListConRanker by introducing listwise encoding through the use of the ListTransformer. The listwise encoding can help learn the global contrastive information between passages. In addition, we use ListAttention to help maintain the features of the query, which assist similarity calculations with passages. Finally, we propose to use the circle loss to replace the cross-entropy loss and solve the problem of data efficiency. Besides, circle loss can smooth the gradient and help the model to find the optimal space. The experimental results on the reranking benchmark of C-MTEB demonstrate the effectiveness of ListConRanker.

6 Limitations

We propose using iterative inference to address the issue of disperse attention when inputting a large number of passages, especially in the MMarcoReranking dataset. This problem arises due to the lack of training samples with large numbers of passages. However, iterative inference leads to the need for multiple inferences on some passages, which increases time complexity. For datasets where each query corresponds to only a small number of passages, the impact of the increased time complexity is subtle. We have other methods to address the issue of large numbers of passages while reducing time complexity. For example, we can sample a small number of passages from the passage set P. We remove these samples from P after obtaining the similarity scores. And we repeat the process multiple times to obtain all similarity scores and get the final order. However, the randomness of sampling can cause instability and uncertainty in the results. Therefore, we need to explore a low-time-complexity inference method for simultaneously inputting a large number of passages in the future.

References

Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. 2016. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Procedings of the British Machine Vision Conference 2016*. British Machine Vision Association.

Luiz Bonifacio, Vitor Jeronymo, Hugo Queiroz Abonizio, Israel Campiotti, Marzieh Fadaee, Roberto Lotufo, and Rodrigo Nogueira. 2021. mmarco: A multilingual version of the ms marco passage ranking dataset. *arXiv preprint arXiv:2108.13897*.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 129–136, New York, NY, USA. Association for Computing Machinery.

Jiayi Chen, Chen Wu, Shaoqun Zhang, Nan Li, Liangjie Zhang, and Qi Zhang. 2024. Efficient ternary weight embedding model: Bridging scalability and performance. *arXiv preprint arXiv:2411.15438*.

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

Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for ir with contextual neural language modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 985–988, New York, NY, USA. Association for Computing Machinery.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. Rethink training of bert rerankers in multi-stage retrieval pipeline. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part II*, page 280–286, Berlin, Heidelberg. Springer-Verlag.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2G: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715, Seattle, United States. Association for Computational Linguistics.

Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. 2020. A deep look into neural ranking models for information retrieval. *Information Processing Management*, 57(6):102067.

Tahmid Hasan, Abhik Bhattacharjee, Md. Saiful Islam, Kazi Mubasshir, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. 2021. XL-sum: Large-scale multilingual abstractive summarization for 44 languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4693–4703, Online. Association for Computational Linguistics.

Junqin Huang, Zhongjie Hu, Zihao Jing, Mengya Gao, and Yichao Wu. 2024. Piccolo2: General text embedding with multi-task hybrid loss training. *Preprint*, arXiv:2405.06932.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Phillip Keung, Yichao Lu, György Szarvas, and Noah A. Smith. 2020. The multilingual Amazon reviews corpus. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*

(EMNLP), pages 4563–4568, Online. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Jianquan Li, Xidong Wang, Xiangbo Wu, Zhiyi Zhang, Xiaolong Xu, Jie Fu, Prayag Tiwari, Xiang Wan, and Benyou Wang. 2023. Huatuo-26m, a large-scale chinese medical qa dataset. *Preprint*, arXiv:2305.01526.

Shiyu Li, Yang Tang, Shizhe Chen, and Xi Chen. 2024. Conan-embedding: General text embedding with more and better negative samples. *Preprint*, arXiv:2408.15710.

Yudong Li, Yuqing Zhang, Zhe Zhao, Linlin Shen, Weijie Liu, Weiquan Mao, and Hui Zhang. 2022. CSL: A large-scale Chinese scientific literature dataset. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3917–3923, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

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 Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew 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. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.

Qi Liu, Bo Wang, Nan Wang, and Jiaxin Mao. 2024. Leveraging passage embeddings for efficient listwise reranking with large language models. *arXiv preprint arXiv:2406.14848*.

Yuxiang Lu, Yiding Liu, Jiaxiang Liu, Yunsheng Shi, Zhengjie Huang, Shikun Feng Yu Sun, Hao Tian, Hua Wu, Shuaiqiang Wang, Dawei Yin, et al. 2022. Ernie-search: Bridging cross-encoder with dual-encoder via self on-the-fly distillation for dense passage retrieval. *arXiv preprint arXiv:2205.09153*.

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

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

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718, Online. Association for Computational Linguistics.

Ramith Padaki, Zhuyun Dai, and Jamie Callan. 2020. Rethinking query expansion for bert reranking. In *Advances in Information Retrieval*, pages 297–304, Cham. Springer International Publishing.

Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. Setrank: Learning a permutation-invariant ranking model for information retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 499–508, New York, NY, USA. Association for Computing Machinery.

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

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2024. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1504–1518, Mexico City, Mexico. Association for Computational Linguistics.

Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3781–3797, Abu Dhabi, United Arab Emirates. 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*, pages 14918–14937, Singapore. Association for Computational Linguistics.

Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. 2020. Circle loss: A unified perspective of pair similarity optimization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6397–6406.

360Zhinao Team. 2024. 360zhinao technical report. *Preprint*, arXiv:2405.13386.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

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*, ICML '08, page 1192–1199, New York, NY, USA. Association for Computing Machinery.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA. Association for Computing Machinery.

Xiaohui Xie, Qian Dong, Bingning Wang, Feiyang Lv, Ting Yao, Weinan Gan, Zhijing Wu, Xiangsheng Li, Haitao Li, Yiqun Liu, and Jin Ma. 2023. T2ranking: A large-scale chinese benchmark for passage ranking. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2681–2690, New York, NY, USA. Association for Computing Machinery.

Sheng Zhang, Xin Zhang, Hui Wang, Jiajun Cheng, Pei Li, and Zhaoyun Ding. 2017. Chinese medical question answer matching using end-to-end character-level multi-scale cnns. *Applied Sciences*, 7(8).

Sheng Zhang, Xin Zhang, Hui Wang, Lixiang Guo, and Shanshan Liu. 2018. Multi-scale attentive interaction networks for chinese medical question answer selection. *IEEE Access*, 6:74061–74071.

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*.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*.

Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. 2021. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China. Chinese Information Processing Society of China.

A How does Circle Loss Smooth the Gradients

In Equations 11-17, we have shown the calculation method of Circle Loss. We show how it smooths the gradient changes during the training process through predicted logits here. It is mainly related to four variables: s, O, α, and m. The γ is a scaling factor and has nothing to do with gradient smoothing. The Δ_{neg} and Δ_{pos} are the between-class and within-class margins respectively, which is similar to the margin value in Triplet Loss.

First, Circle Loss uses m and O to control the upper and lower bounds of the predicted logits. We take the training process of positive passages as an example. In the first-stage training phase, we set $m = -0.2$. When the positive logit is greater than $O_{pos} = 1 + m = 0.8$ in Equation 15, Circle Loss will set α_{pos} through Equation 17. At this time, the loss and gradient of this positive passage is 0.

In addition, when the predicted logit of a positive passage increases from the lower bound to the upper bound, the gradient can also change from large to small through α. This enables the model to accelerate training at the beginning and prevent missing the optimal space by reducing the gradient in the later stage of training. We still take the training of a positive passage as an example. In the early stage of training, when the predicted logit of the positive passage is close to 0, the value of α_{pos} in Equation 17 will be a positive value close to O_{pos}. In the later stage of training, when the predicted logit of the positive passage approaches the upper bound O_{pos}, α_{pos} will be a positive value close to 0. Eventually, α_{pos} will act as a weight for both the loss and the gradient on r_{pos} in Equation 13.

B Comparison between Circle Loss and Other Ranking Loss

In Section 4.4.3 and Table 4, we have demonstrated the differences between Circle Loss and other loss functions (i.e., CoSENT Loss, Triplet Loss, and Cross-Entropy Loss). In addition to these loss functions, there are also some traditional listwise ranking loss functions (e.g., ListNet (Cao et al., 2007) and ListMLE (Xia et al., 2008)). We will compare them in the following.

ListNet and ListMLE are designed for ranking tasks with multiple similarity labels (e.g., A > B > C > D). Only a small number of passages may have the same rank. ListNet and ListMLE can ensure that each passage is finally ranked in the correct position. In this case, the binary cross-entropy (BCE) loss function can not achieve a similar goal. However, since the training and testing data in C-MTEB only have two labels, similar and dissimilar (i.e., A = B = C > D), the training objectives of ListNet and binary cross-entropy loss are quite similar in this case. Their training objectives are both to make the predicted values of positive passages approach 1 and those of negative passages approach 0. And the performance of the BCE loss function can be referred to in Table 4. Finally, Circle Loss is preferred over BCE loss function because it has the property of smoother gradient changes, making it a better choice.

C Dataset Details

We train ListConRanker using the training sets of cMedQA1.0 (Zhang et al., 2017), cMedQA2.0 (Zhang et al., 2018), MMarcoReranking (Bonifacio et al., 2021), T2Reranking (Xie et al., 2023), huatuo(Li et al., 2023), MARC(Keung et al., 2020), XL-sum(Hasan et al., 2021), and CSL(Li et al., 2022). The details of these datasets are presented in Table 5.

- **cMedQA1.0 and cMedQA2.0:** We use the original datasets without any pre-processing.
- **MMarcoReranking:** We use the Simplified Chinese subset of MMarcoReranking. Due to the large number of queries in the Chinese subset (about 40M queries), training on the full dataset risks domain overfitting. Therefore, we randomly selected 400,000 samples from the Chinese subset. Additionally, since each query in the original dataset corresponds to only one positive passage and one negative passage, which differs from the reranking task format, we applied a hard negative mining method to increase the number of negative passages per query to 20.
- **T2Reranking:** Since the T2Reranking dataset has four similarity labels for passages (ranging from 0 to 3, where 0 is the least similar and 3 is the most similar), but the test set is evaluated with only two similarity labels, we treat passages with labels 1 to 3 as positive examples and passages with label 0 as negative examples in the training set. Additionally, we filter out queries that do not have both positive and negative examples, using the remaining queries as the final training set.
- **huatuo and CSL:** Since the original datasets have only one positive passage and no negative passage per query, we applied a hard negative mining method to increase the number of negative passages to 20.
- **MARC and XL-sum:** We use their Simplified Chinese subsets. Similar to huatuo and CSL, since the original datasets have only one positive passage per query and no negative passage, we applied a hard negative mining method to increase the number of negative passages to 20.

Dataset	Number of query	Avg. number of passages per query
cMedQA1.0	5,000	31.87
cMedQA2.0	10,000	51.88
MMarcoReranking	400,000	21.00
T2Reranking	105,668	8.11
huatuo	177,703	21.00
MARC	200,000	21.00
XL-sum	37,362	21.00
CSL	395,927	21.00
Total	1,331,660	20.25

Table 5: The details of all training datasets.

The number of passages obtained through hard negative mining has not been verified through detailed experiments. It was determined empirically.

D The Reason of Two-stages Training

Both the MLPs and ListTransformer are trained from scratch. Since they don't have any meaningful information in the first stage, to prevent the embedding model from being influenced by the MLPs and ListTransformer and learning incorrect information, we freeze the embedding model. Besides, we select a value of m that is easier to learn, so as to enhance the generalization ability of the model and prevent the model from falling into the local optimal space. In the second stage, after the ListTransformer and MLPs have developed semantic discrimination capabilities, we select a more aggressive value of m to enable the ListConRanker to learn domain knowledge.

E Hyperparameters Selections

- l**:** The choice of the l hyperparameter is discussed in Section 4.4.5.
- γ**:** The selection of γ was not specifically tuned but was based on empirical experience.
- θ**:** We set θ to 20 because the average number of passages per query is around 20. The model performs well in terms of ranking when the number of passages is less than or equal to 20, so we decided to stop the iterative reasoning at this point.
- β**:** Setting β to 0.2 is the result of balancing the computational complexity of iterative inference and the ranking performance. Setting it too small would require multiple rounds of iterative inference to complete a single prediction, increasing computational cost. On the other hand, setting it too large would lead to some positive passages being filtered out, causing a decline in model performance.

F Additional Parameters and Computational Complexity

Compared to previous embedding-based baselines (e.g., Conan-embedding-v1, xiaobu-embedding-v2, and zpoint-large-embedding-zh), ListConRanker introduces the ListTransformer module for list-wise encoding. This adds more parameters to the model. The embedding model has 327M parameters. ListConRanker adds an additional 74M parameters (including the ListTransformer and the upper-layer MLPs), bringing the total parameter count to 401M.

However, although a considerable number of parameters have been added, the input sequence length for ListTransformer is small due to the reranking task typically involving only a small number of candidate passages. This is favorable for the $O(N^2)$ computational complexity of attention in ListTransformer. For example, when inputting 1 query and 10 candidate passages, each with a token length of 256, the inference of the embedding model requires 7.18G FLOPS. In this case, the input sequence for ListTransformer has 11 tokens, and it only requires an additional 0.26G FLOPS, which is about a 3.6% increase compared to the embedding model. Therefore, we believe the added ListTransformer module is computationally efficient and worth trying.

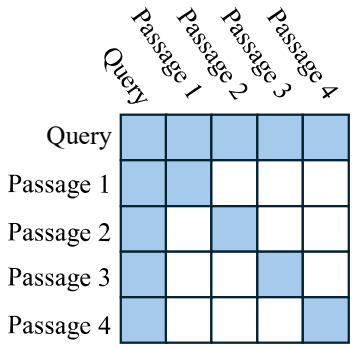

Figure 5: The attention mask of PassageAttention.

G PassageAttention Details

We introduced PassageAttention, an approach opposite to ListAttention, in Section 4.4.1. Its attention mask is shown in Figure 5.

In ListAttention, we allow passages to directly interact through attention and learn comparative information. Each passage learns similarity information from the query and compares itself to other passages. This does not alter the semantic information of the query, ensuring that enough semantic information remains for accurate similarity prediction.

However, the comparison between passages is mediated through the query in PassageAttention, which means the query representation contains information from all the passages. This additional information in query representation is redundant and inefficient for similarity computation. When the query information is lost, the final similarity calculated with the passages results in significant errors, leading to the worst performance.

H Baseline Links

Since most of the baselines do not have technical reports, we mainly provide the links to their open-source models here.

- **Conan-embedding-v1** (Li et al., 2024)**:** https://huggingface.co/TencentBAC/Conan-embedding-v1
- **xiaobu-embedding-v2:** https://huggingface.co/lier007/xiaobu-embedding-v2
- **zpoint-large-embedding-zh:** https://huggingface.co/iampanda/zpoint_large_embedding_zh
- **LdIR-Qwen2-reranker-1.5B:** https://huggingface.co/neofung/LdIR-Qwen2-rer

anker-1.5B

- **ternary-weight-embedding** (Chen et al., 2024)**:** https://huggingface.co/malenia1/ternary-weight-embedding
- **360Zhinao-1.8B-Reranking** (Team, 2024)**:** https://huggingface.co/qihoo360/360Zhinao-1.8B-Reranking
- **piccolo-large-zh-v2** (Huang et al., 2024): **:** https://huggingface.co/sensenova/piccolo-large-zh-v2
- **Yinka:** https://huggingface.co/Classical/Yinka
- **Baichuan-text-embedding:** https://platform.baichuan-ai.com/docs/text-Embedding
- **360Zhinao-search:** https://huggingface.co/qihoo360/360Zhinao-search