LRANKER: LLM RANKER FOR MASSIVE CANDIDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have recently shown strong potential for ranking by capturing semantic relevance and adapting across diverse domains, yet existing methods remain constrained by limited context length and high computational costs, restricting their applicability to real-world scenarios where candidate pools often scale to millions. To address this challenge, we propose `LRanker`, a framework tailored for large-candidate ranking. `LRanker` incorporates a candidate aggregation encoder that leverages K-means clustering to explicitly model global candidate information, and a graph-based test-time scaling mechanism that partitions candidates into subsets, generates multiple query embeddings, and integrates them through an ensemble procedure. By aggregating diverse embeddings instead of relying on a single representation, this mechanism enhances robustness and expressiveness, leading to more accurate ranking over massive candidate pools. We evaluate `LRanker` on seven tasks across three scenarios in *RBench* with different candidate scales. Experimental results show that `LRanker` achieves over *30%* gains in the *RBench-Small* scenario, improves by *3–9%* in MRR in the *RBench-Large* scenario, and sustains scalability with *20–30%* improvements in the *RBench-Ultra* scenario with more than *6.8M* candidates. Ablation studies further verify the effectiveness of its key components. Together, these findings demonstrate the robustness, scalability, and effectiveness of `LRanker` for massive-candidate ranking.

1 INTRODUCTION

Using large language models (LLMs) for ranking has already demonstrated remarkable potential (Li et al., 2023b; Lin et al., 2024; Jiang et al., 2023), showing strong capabilities in capturing semantic relevance, adapting to diverse domains, and achieving competitive performance compared to traditional retrieval and ranking methods. However, constraints such as limited context length (Rashid et al., 2024; Liu et al., 2024b) and prohibitive computational costs (Chen et al., 2025b) restrict current LLM-based ranking methods to small candidate sets, limiting their applicability to real-world scenarios like search and recommendation, where candidate pools often scale to millions. Therefore, our paper aims to raise attention to this pressing research question: *How can we build an efficient LLM ranker for large candidate ranking?*

Existing LLM-based rankers can be broadly distinguished by their input and output formats as shown in Table 1. In terms of input, prior approaches typically adopt one of four strategies: (1) query only (Li et al., 2023a), (2) query combined with a single candidate (Ma et al., 2024), (3) query–candidate pairs (Qin et al., 2023), or (4) the full candidate list (Pradeep et al., 2023; Feng et al., 2025; Sun et al., 2023a). While the last option quickly becomes infeasible due to the limited context length of LLMs, the first three fail to incorporate global candidate-level information, introducing systematic biases into the ranking process. On the output side, most methods directly generate ranking results in the token space, which couples ranking quality with the LLM's decoding latency and restricts scalability.

Based on the above discussion, we argue that *an effective LLM framework for massive-candidate ranking must model global candidate information in the input and perform ranking through embedding-based outputs.* Nevertheless, constructing such LLM rankers faces two key challenges. First, when the number of candidates is large, the limited context length of LLMs makes it difficult to model the global candidate information, which can lead to ranking inaccuracies. Second, relying on a single embedding to rank all candidates limits the expressive capacity of the model, thereby constraining its overall potential.

Table 1: **Comparison of `LRanker` with existing LLM-based rankers across four dimensions: input, output, ranking latency, and maximum candidate scale.** Unlike prior approaches, `LRanker` leverages aggregated candidate centroids within an efficient LLM-based ranking architecture, making its computation independent of candidate size and enabling efficient processing of the information of large-scale candidate sets.

LLM-based Ranker	Input	Output	Ranking Latency	Largest Candidate Scale
PRP (Qin et al., 2023)	Query+ Candidate Pair	Token	High	100
RankGPT (Sun et al., 2023a)	Query + Candidate List	Token	Moderate	100
IRanker (Feng et al., 2025)	Query + Partial Candidate List	Token	Moderate	20
RankLLaMA (Ma et al., 2024)	Query+Single Candidate	Embedding	High	200
`LRanker`	Query + Aggregated Candidate info	Embedding	**Low**	**6.81M**

To address the limitations of existing LLM rankers, we propose `LRanker`, a framework tailored for large-candidate ranking. At the input stage, `LRanker` employs a candidate aggregation encoder that clusters candidate embeddings via K-means and summarizes them into compact centroids, ensuring that global candidate information is explicitly modeled within the prompt. At the inference stage, `LRanker` introduces a graph-based test-time scaling mechanism that iteratively partitions candidates, generates multiple query embeddings under different candidate subsets, and integrates them through an ensemble procedure. This design enriches the representation of the query by aggregating multiple perspectives rather than relying on a single embedding, thereby enhancing robustness and discriminative power for ranking, and enabling more accurate matching across massive candidate pools.

We evaluate `LRanker` on seven tasks across three scenarios in *RBench* with different candidate scales. In the *RBench-Small* setting, `LRanker` achieves over *30%* relative gains compared with existing rankers. In the *RBench-Large* setting, it outperforms existing approaches by about *3–9%* in MRR. Even in the challenging *RBench-Ultra* scenario with more than *6.8M* candidates, `LRanker` sustains scalability and delivers *20–30%* improvements. Ablation studies further confirm that global candidate aggregation, test-time ensemble, and LoRA adaptation all contribute to these gains, demonstrating the robustness of our design.

2 Problem Formulation

Given a query q, the objective of a ranking task (Liu et al., 2009; Li, 2011; Cao et al., 2007) is to train a ranker f that orders a candidate set $D = \{c_1, c_2, \ldots, c_n\}$ of size n. Typically, D can be separated into a positive subset D_p (items that the user truly interacted with, e.g., products actually purchased) and a negative subset D_n (items not chosen). To assess how accurately the ranker retrieves the positives, its performance is evaluated with ranking metrics E, such as Normalized Discounted Cumulative Gain (nDCG) (Järvelin & Kekäläinen, 2002) or Mean Reciprocal Rank (MRR) (Voorhees et al., 1999; Cremonesi et al., 2010).

Formally, a ranker π maps the pair (q, D) into an ordered sequence

$$\pi : (q, D) \mapsto O = \{c_1^{r_1}, c_2^{r_2}, \ldots, c_n^{r_n}\}, \quad O \in \mathbb{S}_n, \tag{1}$$

where r_i denotes the position assigned to candidate c_i, and $\mathbb{S}_n$ is the space of all permutations over n elements. The learning objective is then to identify the optimal ranker π^* within a hypothesis class $\mathcal{F}$ that maximizes the expected evaluation score under the data distribution $\mathcal{Z}$:

$$\pi^* = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{(q,D)\sim\mathcal{Z}} \left[E(\pi(q, D)) \right]. \tag{2}$$

3 LRanker: LLM Ranker for Massive Candidates

Building on the limitations of existing LLM rankers shown in Figure 1, we introduce `LRanker`, a framework designed to handle large-candidate ranking more effectively. `LRanker` improves ranking performance through two complementary components: (1) a candidate aggregation encoder that leverages K-means clustering to capture global candidate information, and (2) a graph-based test-time scaling strategy that integrates query embeddings across multiple candidate scales to enhance inference. We also describe the model training procedure and provide a motivation example to illustrate how these components work together.

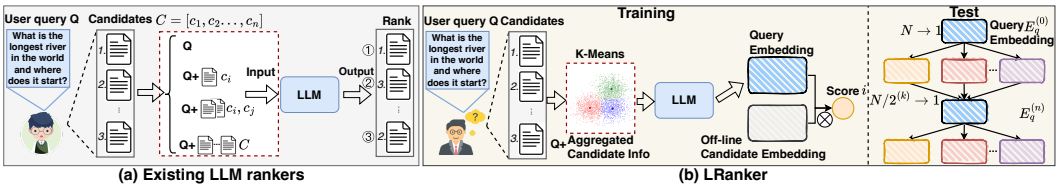

Figure 1: **Compared with existing LLM rankers on large-candidate tasks, `LRanker` incorporates advanced designs in both the representation of candidate information and the inference strategies used during testing.** (a) Existing LLM rankers generally adopt four input formats—a point worth highlighting (see the red box). These include: (1) query only, (2) query with a single candidate, (3) query with candidate pairs, and (4) query with the complete candidate list. The fourth approach is fundamentally constrained by the limited context length of LLMs, rendering it ineffective in massive-candidate scenarios. The first three approaches, in turn, fail to incorporate global candidate-level information, thereby introducing systematic biases into the ranking process. (b)`LRanker` tackles the limitations of existing LLM rankers through two key innovations. First, at the input stage, it employs K-means clustering to construct aggregated candidate info (see the red box), enabling effective modeling of global candidate information. Second, at the testing stage, it introduces a graph-based scaling mechanism that integrates query embeddings across multiple candidate scales, thereby enhancing ranking accuracy and robustness.

3.1 Framework of LRanker

We design `LRanker` as an encoder–decoder framework consisting of two components: a *candidate aggregation encoder* and an *LLM decoder*. The overall goal is to learn query-aware embeddings that integrate both user intent and global candidate information, thereby enabling more accurate passage ranking.

Candidate Aggregation Encoder. To capture the global distributional information of candidates, we apply K-means clustering on their embeddings. Given a candidate set $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ of size N, we obtain:

$$\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_K\} = \text{KMeans}(\{c_1, \ldots, c_N\}), \tag{3}$$

where $\mathcal{G}_k$ represents the k-th cluster. Each cluster centroid is computed as:

$$\mathbf{g}_k = \frac{1}{|\mathcal{G}_k|} \sum_{c_i \in \mathcal{G}_k} \mathbf{e}(c_i), \tag{4}$$

with $\mathbf{e}(c_i)$ denoting the base encoder embedding of candidate c_i. The centroids are then concatenated and projected into the query embedding space:

$$\mathbf{g} = [\mathbf{g}_1; \mathbf{g}_2; \ldots; \mathbf{g}_K], \quad \tilde{\mathbf{g}} = \mathcal{P}(\mathbf{g}). \tag{5}$$

LLM as a Decoder. The LLM serves as a decoder to jointly encode the query and the aggregated candidate information. We design an input prompt that integrates the user query q with the projected aggregated candidate embedding $\tilde{\mathbf{g}}$, as shown in Appendix A. A special placeholder token `<|embedding|>` is reserved in the prompt and replaced with $\tilde{\mathbf{g}}$, allowing the LLM to condition its representation on the global candidate context.

For a query q with T tokens, let $\mathbf{z}_{q,t}$ denote the hidden state of the t-th token and $\mathbf{z}_{q,\text{nt}}$ the hidden state of the predicted next token. The query embedding is obtained by averaging over all token states:

$$\mathbf{h}_q = \frac{1}{T+1} \left(\mathbf{z}_{q,\text{nt}} + \sum_{t=1}^{T} \mathbf{z}_{q,t} \right). \tag{6}$$

Similarly, for each candidate c_i with length $|c_i|$, the off-line candidate embedding is:

$$\mathbf{h}_{c_i} = \frac{1}{|c_i|+1} \left(\mathbf{z}_{c_i,\text{nt}} + \sum_{j=1}^{|c_i|} \mathbf{z}_{c_i,j} \right), \tag{7}$$

Table 2: **Detailed summarization of tasks used in our ranking tasks with varying candidate scales.** We summarize the scenarios, task names, candidate sizes, and the number of queries.

Scenario	Task	Candidate Size	# Query Num
RBench-Small			
Rec-Music	Recommendation	20	12,483
Routing-Balance	Routing	20	1,620
RBench-Large			
Rec-Movie	Recommendation	3,884	4,167
Rec-Toy	Recommendation	11,925	19,413
MS MARCO	Passage ranking	24,697	3,038
ESCI	Product searching	4,000	3,999
RBench-Ultra			
Rec-Clothing	Recommendation	6,805,462	185,925

where $\mathbf{z}_{c_i,j}$ and $\mathbf{z}_{c_i,\text{nt}}$ are the hidden states of the j-th token and predicted next token, respectively. Finally, we compute relevance scores and rank the candidates:

$$s(q, c_i) = \langle \mathbf{h}_q, \mathbf{h}_{c_i} \rangle, \quad \pi(q) = \text{argsort}\big(\{s(q, c_i)\}_{i=1}^{N}\big), \tag{8}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product and $\pi(q)$ represents the ordered sequence of candidates.

3.2 Training LRanker

During training, we adopt a ranking objective with both positive and negative samples. For each query q, let c^+ denote the ground-truth relevant candidate and $\mathcal{C}^-$ the set of sampled negative candidates. The model is trained to assign a higher score to c^+ than to any negative candidate $c^- \in \mathcal{C}^-$. Concretely, the loss function is defined as a softmax cross-entropy:

$$\mathcal{L} = -\log \frac{\exp(s(q, c^+))}{\exp(s(q, c^+)) + \sum_{c^- \in \mathcal{C}^-} \exp(s(q, c^-))}, \tag{9}$$

where $s(q, c)$ is the relevance score between query q and candidate c.

To improve the robustness of the aggregated candidate representation, we further introduce a *random partition sampling* strategy during training. Specifically, for each training instance, the candidate set $\mathcal{C}$ is randomly split into two disjoint partitions. We then compute aggregated candidate info based on one of the partitions and update the model accordingly. This strategy enables the encoder to learn from diverse candidate scales and prepares the model to better support test-time scaling, where multiple candidate partitions of varying sizes are integrated for final ranking.

3.3 Graph-Based Test-Time Scaling

Motivated by the principle of ensemble learning (Zhou, 2025; Breiman, 1996; Freund & Schapire, 1997; Wolpert, 1992)—where combining multiple classifiers outperforms relying on a single one—we extend this idea to ranking by aggregating multiple query embeddings. The key intuition is that a single query embedding may be biased by the initial candidate context, while combining embeddings obtained under diverse candidate partitions can lead to more robust ranking performance.

Concretely, for a ranking task with N candidates, we first obtain an initial query embedding $E_q^{(0)}$ using the proposed encoder–decoder framework as shown in Figure 1(b). Based on $E_q^{(0)}$, we perform an elimination step: the candidates are partitioned into k subsets of size $N/2^{(k)}$, and only the top-ranked candidates within each subset are retained. For each subset, we update the aggregated candidate information and compute a new query embedding. This yields k embeddings, which are then averaged with the original $E_q^{(0)}$ to produce an updated embedding $E_q^{(1)}$:

$$E_q^{(1)} = \frac{1}{k+1} \left(E_q^{(0)} + \sum_{m=1}^{k} E_{q,m}^{(0)} \right), \tag{10}$$

where $E^{(0)}_{q,m}$ denotes the embedding obtained from the m-th partition in the first elimination round.

This elimination-and-update process can be repeated for multiple iterations, producing a sequence of embeddings $E^{(0)}_q, E^{(1)}_q, \ldots, E^{(i)}_q$. At test time, the final ranking score for a candidate c is computed by averaging its scores across all embeddings in the sequence:

$$s_{\text{final}}(q, c) = \frac{1}{i+1} \sum_{t=0}^{i} s_{E^{(t)}_q}(q, c), \tag{11}$$

where $s_{E^{(t)}_q}(q, c)$ is the score computed with embedding $E^{(t)}_q$.

We refer to k (the number of partitions per elimination step) as the *width* of test-time scaling, and to i (the number of embedding update iterations) as its *depth*. Together, this forms a graph-based scaling procedure, where embeddings propagate along a graph of candidate partitions to refine query representations iteratively. In practice, both width k and depth i are selected based on validation performance, and the best hyperparameters are directly applied at inference time.

3.4 Qualitative Illustration

We provide a qualitative illustration in Figure 2 to show how `LRanker` 's test-time scaling mechanism leverages multiple query embeddings to enhance representation quality compared to using a single embedding in a dual-tower setup. On a recommendation dataset with 100 candidates, we apply t-SNE to visualize candidate embeddings, the ground-truth item embedding, and the query embeddings produced at different $N \to 1$ scales. The plots indicate that the averaged query embedding from test-time ensemble integrates complementary strengths from individual embeddings and qualitatively appears closer to the ground-truth, providing intuitive evidence for its potential to improve ranking performance.

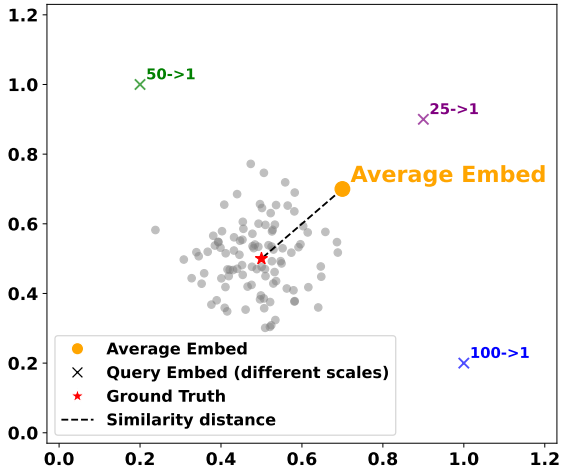

Figure 2: **The graph-based test-time ensemble mechanism qualitatively illustrates how combining multiple embeddings can provide richer representations than relying on a single query embedding.** On a recommendation dataset with 100 candidates, we use t-SNE to visualize candidate embeddings (gray points), the ground-truth item embedding (red star), and query embeddings produced by `LRanker` at different $N \to 1$ scales. The visualizations suggest that the averaged query embedding obtained through test-time ensemble captures complementary information from multiple embeddings and tends to lie closer to the ground-truth embedding.

4 Experiments

To explore the capability of LLMs for large-candidate ranking, we conduct a comprehensive training and evaluation of the proposed `LRanker` across seven interdisciplinary tasks in LLM ranking bench (RBench) with varying candidate scales. We then compare its performance against both general ranking baselines and domain-specific methods. We begin by introducing the tasks within the LLM ranking framework.

Task description. The details of the tasks are summarized across three scenarios in Table 2.

- **RBench-Large.** For the large-candidate ranking scenario, we employ four widely used datasets, where the number of candidates ranges from several thousand to over ten thousand as shown in Table 2. We begin our experiments with two sequential recommendation datasets, MovieLens ml-1m (Rec-Movie) (Hou et al., 2024a) and Amazon Toys (Rec-Toy) (McAuley et al., 2015; Ni et al., 2019). For both tasks, following prior studies (Geng et al., 2022; Hua et al., 2023), we construct each sample by extracting 20 consecutive interactions as the historical sequence, while

designating the 21st interaction as the ground-truth item. For evaluation, we adopt the widely used leave-one-out strategy. In addition, we adopt datasets from passage ranking and product search tasks, namely MS MARCO (Bajaj et al., 2016) and ESCI (Reddy et al., 2022), respectively. For both datasets, we assign one positive sample to each query and construct the negative sample set for each query by aggregating the negative samples from all queries.

- **RBench-Ultra.** This scenario is designed to investigate the capability limits of the LLM-based `LRanker` in addressing ultra-large ranking tasks with candidate pools at the million scale. To this end, we employ the sequential recommendation dataset Amazon Clothing (Rec-Clothing) (McAuley et al., 2015; Ni et al., 2019), which contains nearly 7 million candidate items as shown in Table 2. For this task, we follow the same setting as in the RBench-Large setup, namely extracting 20 consecutive interactions as the historical sequence and designating the 21st interaction as the ground-truth item. Evaluation is also conducted using the widely adopted leave-one-out strategy.
- **RBench-Small.** We design this scenario to explore the capability of `LRanker` in ranking scenarios with relatively small candidate sets, such as the re-ranking stage in recommender systems. To this end, following prior work (Feng et al., 2025; 2024), we adopt the sequential recommendation task Rec-Music and the LLM routing task Routing-Balance, both with 20 candidates as shown in Table 2, which is fully consistent with the setting in (Feng et al., 2025).

Baselines and metrics. We evaluate a variety of baseline methods across three scenarios. The baselines are categorized into two groups: **(a)** ***General baselines*** that apply across tasks, and **(b)** ***Task-specific baselines*** tailored to each task. For all methods, we primarily use Mean Reciprocal Rank (MRR) (Voorhees et al., 1999; Cremonesi et al., 2010) and Normalized Discounted Cumulative Gain (NDCG@K) (Järvelin & Kekäläinen, 2002; Burges et al., 2005; Liu et al., 2009) with K = 10 to evaluate ranking performance in the main text.

- **General baselines.** We consider two categories of general baselines: retrieval-based methods and LLM-based methods. In retrieval-based methods, user queries or histories are treated as the query, while candidates are regarded as the corpus. We employ both a classical probabilistic retrieval model and a modern dense retrieval model: 1) *BM25* (Robertson et al., 2009), a traditional probabilistic retrieval function that leverages term frequency, inverse document frequency, and document length normalization. 2) *Contriever* (Izacard et al., 2021), a state-of-the-art dense retrieval model trained with contrastive learning and hard negatives.
- **Task-specific baselines.** For the **recommendation tasks in scenarios of RBench-Large and RBench-Ultra**, following prior work on large-scale recommendation (Rajput et al., 2023), we implemented five sequential recommendation baselines: 1) *FM* (Rendle, 2010): A general predictive model that efficiently captures all pairwise feature interactions, widely used as a strong baseline for recommendation and CTR prediction. 2) *BERT4Rec* (Sun et al., 2019): A sequential recommender that applies the bidirectional Transformer (BERT) architecture to user–item sequences, enabling effective modeling of complex item dependencies. 3) *GRU4Rec* (Hidasi et al., 2015): A session-based recommendation model that leverages gated recurrent units (GRUs) to capture sequential dependencies in user interaction data. 4) *SASRec* (Kang & McAuley, 2018): A Transformer-based sequential recommender that employs self-attention to model both short- and long-term user preferences. 5) *Tiger* (Rajput et al., 2023): A state-of-the-art generative retrieval framework designed for large-scale recommendation, which encodes items into semantic IDs and autoregressively generates them for efficient ranking under massive candidate sets.

  For the **recommendation tasks in RBench-Small scenario**, we follow the baseline setup in (Feng et al., 2025) and adopt three representative methods: 1) *SASRec* (Kang & McAuley, 2018): A self-attention-based sequential recommender that models users' sequential behavioral patterns using a Transformer architecture. 2) *BPR* (Rendle et al., 2012): A pairwise ranking method that optimizes sequential recommendation by encouraging observed items to be ranked higher than unobserved ones. 3) *R1-Rec* (Lin et al., 2025): A reinforcement learning-based framework that directly optimizes retrieval-augmented LLMs for recommendation tasks using downstream feedback. As for the **routing task**, we compared three representative routers: 1) *RouterKNN* (Hu et al., 2024): A simple yet effective routing baseline that assigns queries to models by retrieving similar examples and applying majority voting. 2) *RouterBERT* (Ong et al., 2024): A lightweight BERT model fine-tuned for routing decisions using classification over task labels. 3) *GraphRouter* (Feng et al., 2024): A state-of-the-art graph-based router that balances performance and cost through structural modeling.

Table 3: **Model performance comparison with general ranking baselines and task-specific baselines across four scenarios on NDCG@10 and MRR**. Left: Rec-Movie and Rec-Toy. Right: MS MARCO and ESCI. **Bold** and underline denote the best and second-best results.

Model	Rec-Movie		Rec-Toy	
	NDCG@10	MRR	NDCG@10	MRR
General Ranking Baselines				
BM25	0.18	0.54	0.37	0.42
Contriever	0.24	0.43	0.84	1.11
Task-specific Baselines				
FM	2.35	2.01	0.95	0.98
BERT4Rec	4.08	3.56	1.26	1.31
GRU4Rec	4.12	3.59	1.59	1.46
SASRec	4.36	3.84	1.65	1.52
Tiger	7.37	6.12	2.99	2.33
LRanker	**8.02**	**7.80**	**3.21**	**2.42**

Model	MS MARCO		ESCI	
	NDCG@10	MRR	NDCG@10	MRR
General Ranking Baselines				
BM25	34.77	26.28	33.70	23.77
Contriever	44.36	33.29	29.41	26.17
Task-specific Baselines				
RankBERT-110M	42.26	28.59	42.39	31.45
Multilingual-E5-560M	53.73	46.49	53.17	48.43
KaLM-mini-instruct-0.5B	50.57	40.07	55.21	50.28
BGE-Rerank-v2-m3-568M	53.43	47.74	52.02	48.10
RankLLaMA 8B	52.22	48.83	55.78	52.37
LRanker	**54.80**	**49.28**	**58.80**	**57.01**

Finally, for the **passage ranking and product search tasks**, we implemented three specialized ranking baselines: 1) *RankBERT-110M* (Nogueira & Cho, 2019): A BERT-based passage reranker fine-tuned on MS MARCO relevance judgments, treating ranking as a binary classification problem. 2) *Multilingual-E5-560M* (Wang et al., 2022): A multilingual embedding model optimized for retrieval and ranking tasks, trained with contrastive learning objectives to generate semantically meaningful embeddings across languages. 3) *KaLM-mini-instruct-0.5B* (Hu et al., 2025): A 0.5B-parameter multilingual embedding model, instruction-tuned for retrieval and ranking tasks. 4) *BGE-Rerank-v2-m3-568M* (Xiao et al., 2024): A state-of-the-art reranker from the BGE series, fine-tuned on large-scale relevance datasets to enhance cross-encoder-based ranking performance. 5) *RankLLama-8B* (Ma et al., 2024): A ranking-specialized version of Llama-2-8B fine-tuned for passage ranking using pairwise and listwise objectives.

Implementation details. We implement LRanker on top of Qwen3-0.6B embedding[1] with LoRA adaptation. The base encoder produces 1024-dimensional embeddings, which are fused with global candidate cluster features. Candidate clusters are projected using a linear–BatchNorm–ReLU block to 512 dimensions, while user histories are aggregated through a 2-layer MLP (512 hidden units) with positional embeddings and mean pooling. The concatenated representation forms a 1536-dimensional vector, which is aligned with the textual embedding space for retrieval. We adopt InfoNCE loss with temperature 0.15, contrasting positives against sampled negatives. The model is trained for 15 epochs with the AdamW optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay$= 0.01$). The learning rate is set to 1×10^{-4} with a linear warm-up over the first 10% of steps followed by cosine decay. The batch size is 20. LoRA is applied to attention and feed-forward layers with rank=32, $\alpha = 64$, and dropout=0.1. To improve efficiency and stability, we enable BF16 training, gradient checkpointing, and gradient clipping (norm 0.5). For evaluation, we determine the best graph depth and width using the validation set, and fix these configurations when testing on the held-out test set. All experiments are conducted on a single NVIDIA A6000 GPU.

4.1 LRanker Outperforms General Ranking Methods and Task-Specific Baselines

In the RBench-Large scenario, we compare the **0.6B-sized** LRanker with both general ranking baselines and task-specific baselines across four tasks—Rec-Movie, Rec-Toy, MS MARCO, and ESCI—as shown in Table 4. Across all four tasks, LRanker consistently outperforms the strongest existing baselines by relative margins ranging from about 3% to nearly 9% in MRR, establishing clear SoTA performance. In the recommendation setting (Rec-Movie, Rec-Toy), where specialized sequential models such as Tiger dominate, LRanker still secures 7–9% relative improvements, showing that its centroid-based design provides complementary advantages even when strong temporal signals are available. In the retrieval setting (MS MARCO, ESCI), where large-scale candidate pools pose significant efficiency and quality challenges, LRanker achieves 3–9% relative gains over the best LLM rerankers (e.g., RankLLaMA, BGE-Rerank). Notably, the larger improvement on ESCI highlights LRanker's robustness in multilingual and noisy e-commerce search scenarios. Together, these results confirm that LRanker not only scales effectively across candidate sizes but also generalizes well across both recommendation and retrieval domains, consistently surpassing both traditional IR methods and task-specific LLM-based rankers.

[1] https://huggingface.co/Qwen/Qwen3-0.6B

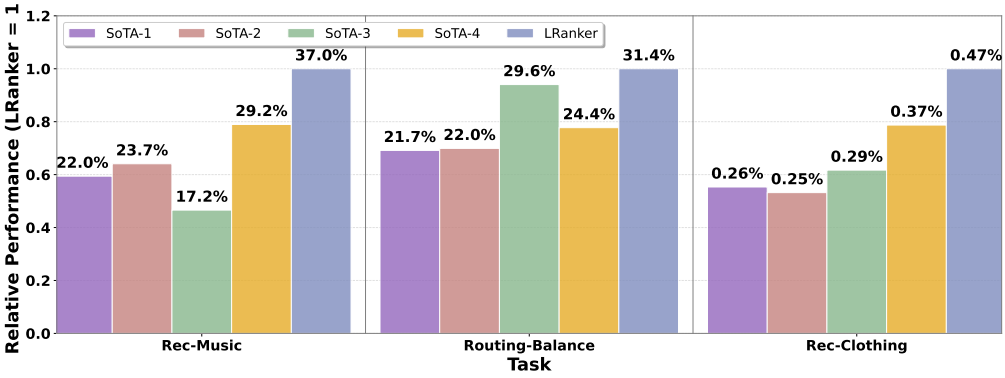

Figure 3: **Compared with state-of-the-art domain-specific baselines, `LRanker` consistently outperforms them across both ultra-long and ultra-short scenarios.** We compared the performance of `LRanker` against four representative SOTA methods across three tasks. Among them, Rec-Music and Routing-Balance are tasks in the RBench-Small scenario, while Rec-Clothing is a task in the RBench-Ultra scenario. Specifically, SOTA-1, SOTA-2, SOTA-3, and SOTA-4 correspond to SASRec (Kang & McAuley, 2018), BPR (Rendle et al., 2012), R1-Rec (Lin et al., 2025), and IRanker (Feng et al., 2025) in the Rec-Music task; GraphRouter (Feng et al., 2024), RouterBert (Ong et al., 2024), RouterKNN (Hu et al., 2024), and IRanker (Feng et al., 2025) in the Routing-Balance task; BERT4Rec (Kang & McAuley, 2018), GRU4Rec (Hidasi et al., 2015), SASRec (Nogueira et al., 2020), and Tiger (Rajput et al., 2023) in the Rec-Clothing task.

4.2 LRanker Achieves Superior Results in Both RBench-Ultra and RBench-Small Scenarios

In the RBench-Small and RBench-Ultra scenarios, we compare the 0.6B-sized `LRanker` with representative state-of-the-art domain-specific baselines across three tasks—Rec-Music, Routing-Balance, and Rec-Clothing—as illustrated in Figure 3. `LRanker` consistently establishes superior performance over all baselines. On RBench-Small, `LRanker` achieves substantial relative improvements, with gains of up to 37% on Rec-Music and over 30% on Routing-Balance compared with the strongest sequential and routing-specific baselines. These results highlight that even in short-context ranking settings with small candidate pools, `LRanker` delivers clear benefits beyond specialized architectures such as SASRec, IRanker, and GraphRouter. On RBench-Ultra, where Rec-Clothing involves over 6.8M candidates, `LRanker` still surpasses highly optimized sequential recommenders (e.g., BERT4Rec, GRU4Rec, Tiger) by 20–30% in relative performance, underscoring its scalability to extreme candidate sizes. Overall, these findings confirm that `LRanker` not only excels in ultra-short candidate scenarios but also scales effectively to ultra-large tasks, demonstrating versatility across diverse ranking regimes.

4.3 Ablation Studies Confirm the Effectiveness of LRanker 's Key Components

To provide a comprehensive understanding of the key components of `LRanker`, we conduct a series of experiments to investigate the effect of different components.

- **w/o global info**: Evaluates the contribution of incorporating global candidate information. This variant removes the clustered embedding input and its associated projector from the `LRanker` framework.
- **w/o test-time ensemble**: Assesses the impact of the test-time ensemble mechanism. In this setting, `LRanker` performs ranking solely using the initial embedding generated by the LLM.
- **w/o LoRA**: Examines the role of LoRA-based training. Here, the LLM parameters are frozen during training, and only the projector is fine-tuned.

We report the evaluation results on Rec-Movie, Rec-Toy, MS MARCO, and ESCI dataset in Figure 4. It can be observed that removing global candidate information causes a clear degradation across all tasks. Without clustered embeddings and the corresponding projector, the model fails to capture global context, which limits its ability to discriminate among candidates and lowers ranking accuracy. Removing the test-time ensemble mechanism also leads to reduced performance. Without the ensemble, the model relies solely on a single embedding from the LLM, which weakens its adaptability

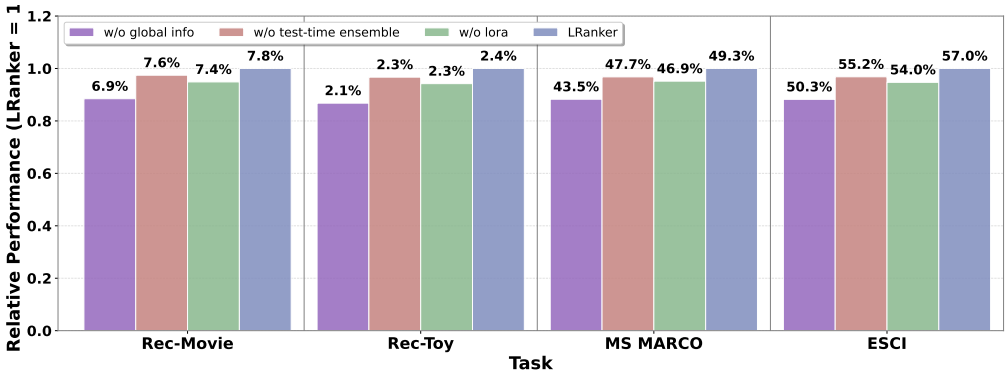

Figure 4: **Ablation studies confirm that each component of `LRanker` contributes positively to the overall performance.** To further examine their roles, we evaluate three ablated settings: (i) w/o global info removes aggregated candidate information, excluding the clustered embedding input and its projector; (ii) w/o test-time ensemble disables the ensemble mechanism, relying only on the initial embedding from the LLM; and (iii) w/o LoRA freezes LLM parameters during training and only fine-tunes the projector. As shown across Rec-Movie, Rec-Toy, MS MARCO, and ESCI, removing any component consistently leads to performance degradation.

to task-specific variations and reduces robustness. Nevertheless, the results without this mechanism remain reasonably strong, suggesting that the ensemble mainly serves as a performance booster. This indicates that while test-time ensemble can further improve ranking quality, users who prioritize inference efficiency may choose to omit it with only a modest loss in performance. This highlights the flexibility of LRanker in accommodating different application needs. Eliminating LoRA-based training likewise results in a performance drop. Freezing the LLM parameters and only fine-tuning the projector prevents the model from learning task-specific adaptations, making it harder to fully exploit the LLM's representational capacity.

5 Additional Related Work

Recent works have explored leveraging large language models (LLMs) for ranking under different paradigms. Token-space ranking methods treat LLMs as text rankers by converting queries and candidates into textual prompts, either through iterative elimination (IRanker (Feng et al., 2025), PRP (Qin et al., 2023)) or one-shot generation (DRanker (Feng et al., 2025), RankVicuna (Pradeep et al., 2023)). However, these approaches face efficiency and context length limitations for large candidate sets. Embedding-based paradigms address this: single-tower methods (RankLLaMA (Ma et al., 2024)) use neural scoring heads but require independent candidate scoring, while dual-tower methods improve efficiency through pre-computed embeddings but limit expressiveness with single query embeddings. Generative LLM approaches have shown strong performance across diverse ranking tasks (Liu et al., 2024a; Sun et al., 2023b; Yoon et al., 2024; Chen et al., 2025a; Hou et al., 2024b). Prompting-based methods (PRP (Qin et al., 2023), LLM4Rec (Hou et al., 2024a)) leverage LLM generalization with minimal modification, while instruction tuning approaches (GPT4Rec (Li et al., 2023a), RankRAG (Yu et al., 2024)) fine-tune models for domain-specific ranking signals. These works highlight LLMs' potential as general-purpose rankers while exposing limitations in efficiency, scalability, and complex reasoning.

6 Conclusion

We propose `LRanker`, a framework designed to address the challenges of large-candidate ranking with LLMs by integrating candidate aggregation and graph-based test-time scaling. Extensive experiments across three scenarios in *RBench* demonstrate that `LRanker` consistently outperforms existing approaches, achieving substantial improvements from small-scale to million-level candidate pools. Ablation studies further validate the effectiveness of its key components. In future work, we plan to extend `LRanker` to a broader range of ranking tasks, further exploring its generality and applicability in real-world settings.

Ethics Statement

All authors of this paper have read and adhered to the ICLR Code of Ethics. Our work does not involve human subjects, personal data, or sensitive attributes. We followed best practices for data usage, ensured compliance with licensing terms, and considered potential risks of bias or misuse.

Reproducibility Statement

We have made every effort to ensure the reproducibility of our results. Details of the model architecture, training settings, and hyperparameters are described in Section 4. All datasets we used are publicly available. The training scripts and evaluation code will be released upon publication to facilitate replication.

References

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

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, 2005.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, 2007.

Yang Chen, Min Zhang, Yiqun Wu, and Yanyan Liu. Rank-r1: Enhancing reasoning in llm-based document reranking. *arXiv preprint arXiv:2503.06034*, 2025a.

Yiqun Chen, Qi Liu, Yi Zhang, Weiwei Sun, Xinyu Ma, Wei Yang, Daiting Shi, Jiaxin Mao, and Dawei Yin. Tourrank: Utilizing large language models for documents ranking with a tournament-inspired strategy. In *Proceedings of the ACM on Web Conference 2025*, pp. 1638–1652, 2025b.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pp. 39–46. ACM, 2010.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834*, 2024.

Tao Feng, Zhigang Hua, Zijie Lei, Yan Xie, Shuang Yang, Bo Long, and Jiaxuan You. Iranker: Towards ranking foundation model. *arXiv preprint arXiv:2506.21638*, 2025.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM conference on recommender systems*, pp. 299–315, 2022.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems. In *European Conference on Information Retrieval*, pp. 364–381. Springer, 2024a.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. Large language models are zero-shot rankers for recommender systems. In *Advances in Information Retrieval: 46th European Conference on IR Research, ECIR 2024, Glasgow, UK, March 24–28, 2024, Proceedings, Part II*, volume 14685 of *Lecture Notes in Computer Science*, pp. 364–381. Springer, 2024b. doi: 10.1007/978-3-031-56060-6_24. URL https://arxiv.org/abs/2305.08845.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Xinshuo Hu, Zifei Shan, Xinping Zhao, Zetian Sun, Zhenyu Liu, Dongfang Li, Shaolin Ye, Xinyuan Wei, Qian Chen, Baotian Hu, et al. Kalm-embedding: Superior training data brings a stronger embedding model. *arXiv preprint arXiv:2501.01028*, 2025.

Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. How to index item ids for recommendation foundation models. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, pp. 195–204, 2023.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pp. 197–206. IEEE, 2018.

Hang Li. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

Jinming Li, Wentao Zhang, Tian Wang, Guanglei Xiong, Alan Lu, and Gerard Medioni. Gpt4rec: A generative framework for personalized recommendation and user interests interpretation. *arXiv preprint arXiv:2304.03879*, 2023a.

Lei Li, Yongfeng Zhang, and Li Chen. Prompt distillation for efficient llm-based recommendation. In *Proceedings of the 32nd ACM international conference on information and knowledge management*, pp. 1348–1357, 2023b.

Jiacheng Lin, Tian Wang, and Kun Qian. Rec-r1: Bridging generative large language models and user-centric recommendation systems via reinforcement learning. *arXiv preprint arXiv:2503.24289*, 2025.

Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 365–374, 2024.

Qidong Liu, Xian Wu, Wanyu Wang, et al. Llmemb: Large language model can be a good embedding generator for sequential recommendation. *arXiv preprint arXiv:2409.19925*, 2024a.

Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

Wenhan Liu, Xinyu Ma, Yutao Zhu, Ziliang Zhao, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. Sliding windows are not the end: Exploring full ranking with long-context large language models. *arXiv preprint arXiv:2412.14574*, 2024b.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2421–2425, 2024.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 188–197, 2019.

Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. URL https://arxiv.org/abs/1901.04085.

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

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*, 2024.

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

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*, 2023.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36:10299–10315, 2023.

Muhammad Shihab Rashid, Jannat Ara Meem, Yue Dong, and Vagelis Hristidis. Ecorank: Budget-constrained text re-ranking using large language models. *arXiv preprint arXiv:2402.10866*, 2024.

Chandan K Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. Shopping queries dataset: A large-scale esci benchmark for improving product search. *arXiv preprint arXiv:2206.06588*, 2022.

Steffen Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pp. 995–1000. IEEE, 2010.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is chatgpt good at search? investigating large language models as re-ranking agents. *arXiv preprint arXiv:2304.09542*, 2023a.

Yixin Sun, Yiqun Zhang, Jiaxin Ma, Yanyan Liu, Yanyan Shao, and Shaoping Zhou. Rankgpt: Enhancing zero-shot ranking with instruction-finetuned large language models. *arXiv preprint arXiv:2304.09542*, 2023b.

Ellen M Voorhees et al. The trec-8 question answering track report. In *Trec*, volume 99, pp. 77–82, 1999.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 641–649, 2024.

Jinhyuk Yoon, Minbyul Jeong, Chan Kim, and Minjoon Seo. Listt5: Listwise reranking with fusion-in-decoder. *arXiv preprint arXiv:2402.15838*, 2024.

Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. Rankrag: Unifying context ranking with retrieval-augmented generation in llms. *Advances in Neural Information Processing Systems*, 37:121156–121184, 2024.

Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2025.

A PROMPT USAGE

To unify the input format across different tasks, we design prompt templates that integrate the user query with the aggregated candidate information through the special token <|embedding|>. These templates guide the model to attend not only to the query but also to the global context of candidate representations. Specifically, we construct task-specific templates for four representative tasks: Recommendation (Table 4), Routing (Table 5), Passage Ranking (Table 6), and Product Searching (Table 7). Each template follows a unified structure but adapts the final instruction to match the objective of the corresponding task.

Table 4: **Prompt template for Recommendation task.**

Task: Recommendation Query: [USER QUERY] <|embedding|> Based on the global context information (candidate items) and the query above, recommend the most relevant item.

Table 5: **Prompt template for Routing task.**

Task: Routing Query: [USER QUERY] <|embedding|> Based on the global context information (candidate LLMs/agents) and the query above, select the most suitable route or model.

Table 6: **Prompt template for Passage Ranking task.**

Task: Passage Ranking Query: [USER QUERY] <|embedding|> Based on the global context information (candidate passages) and the query above, identify the most relevant passage.

B LLM WRITING USAGE DISCLOSURE

An LLM was applied as a writing aid to enhance the clarity and linguistic quality of this paper, specifically by correcting grammatical errors and polishing sentence flow. No part of the research design, data analysis, or interpretation relied on the use of the LLM.

Table 7: **Prompt template for Product Searching task.**

Task: Product Searching Query: [USER QUERY] <|embedding|> Based on the global context information (candidate products) and the query above, return the product that best matches the search intent.