WeRank: Scaling Listwise Reranking to Hundreds of Long Documents

Anonymous authors
Paper under double-blind review

Abstract

Large language models (LLMs) make listwise reranking appealing by enabling multiple candidates to be compared jointly under a shared query context. However, existing LLM-based rerankers are primarily designed for short passages and modest candidate sets, leaving listwise reranking over hundreds of long documents largely underexplored. To study this practically important regime, we introduce *LoDoc*, a real-world long-document reranking benchmark derived from production search traffic. LoDoc contains 14,884 queries, with 206.9 candidate documents per query and 2,081.7 tokens per document on average. We further propose *WeRank*, a scalable listwise reranking framework built on structured long-document compression. WeRank performs fine-grained chunk-level condensation and connects localized representations using a chained condensed chunk context scheme, enabling cross-region context modeling through compact memory states. The resulting representations enable hundreds of long-document candidates to be jointly compared within a shared ranking context. On the human-verified LoDoc test set, WeRank achieves 0.486 nDCG@10 and remains effective as the candidate pool scales to 300 documents. Under a matched representation budget, our structured compression design improves nDCG@10 from 0.400 with document-level compression to 0.443, a 10.8% relative gain. Experiments further demonstrate favorable inference efficiency and validate the key design choices of WeRank. *We will make the dataset and source code publicly available*.

1 Introduction

Large language models (LLMs) have recently emerged as a powerful paradigm for document reranking (Zhi et al., 2026), benefiting from their strong language understanding and reasoning capabilities. Among different formulations, listwise reranking (Liu et al., 2025a) is particularly appealing because it allows multiple candidate documents to be compared jointly under a shared query context rather than scored independently, naturally modeling the relative nature of ranking.

Despite this appeal, existing listwise rerankers are primarily developed for relatively short passages and modest candidate sets. Real-world retrieval systems can operate in a substantially more demanding regime, where each query is associated with hundreds of candidates and each document may contain thousands of tokens (Jiang et al., 2020). This gives rise to an important but underexplored setting: listwise reranking over hundreds of long documents. In this regime, document length and candidate-set size scale simultaneously, leading to hundreds of thousands of candidate-document tokens per query and substantial query-time computation even when the input can technically fit within a long context window.

Existing approaches mainly reduce either the amount of document content exposed to the reranker or the number of candidates compared at once. As illustrated in Fig. 1, one common strategy is to operate on short passages or truncate document content (Liu et al., 2025b), which reduces the per-candidate context burden but becomes increasingly restrictive when relevance evidence is distributed across different regions of a long document. Another widely used strategy is sliding-window reranking (Sun et al., 2023; Pradeep et al., 2023a; Zhi et al., 2026), where only a subset of candidates is compared in each model invocation and the final ranking is progressively constructed across multiple overlapping windows. While effective in controlling per-call context size, this strategy replaces a single global comparison with a sequence of local ranking decisions and repeatedly processes overlapping candidates as the candidate pool grows.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

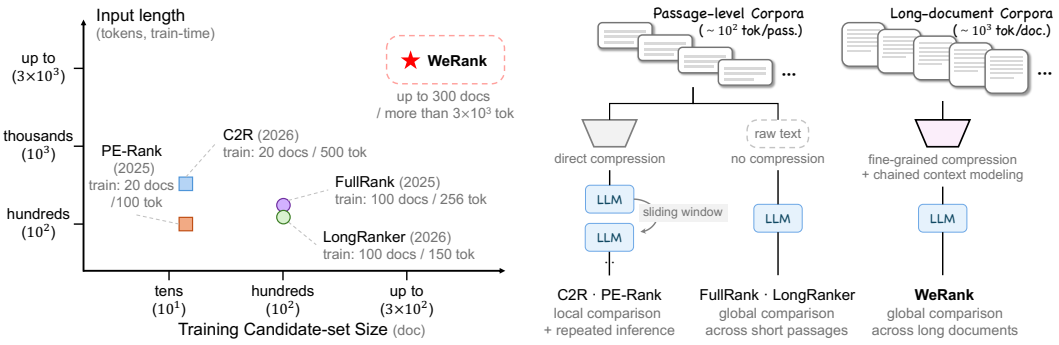


Figure 1: **Comparison of representative listwise rerankers.** (a) Candidate-set size versus per-document input length at training time. Existing methods are mainly trained on short passages and/or modest candidate sets, whereas WeRank scales to hundreds of candidates with thousand-token documents. (b) Comparison of how candidate documents are represented and compared during listwise reranking. Existing methods use raw short passages, direct document compression, or sliding-window inference, while WeRank combines fine-grained chunk-level compression with chained context modeling to enable global comparison across long documents.

More recently, compression-based reranking provides a promising alternative by mapping documents into compact learned representations, allowing more candidates to participate in listwise comparison (Liu et al., 2025a; Zhi et al., 2026). Yet existing compression-based rerankers largely treat each candidate as a whole, which becomes increasingly restrictive for long documents where relevant evidence may be distributed across multiple regions and depend on broader document context. This raises a central challenge: *how can long documents be compactly represented while modeling context across fine-grained document regions for large-scale listwise reranking?*

To address this challenge, we present *WeRank*, a scalable framework built on structured long-document compression for listwise reranking over hundreds of long documents. WeRank performs Chunk-Level Document Condensation to preserve information across local document regions, and connects these representations through Chained Condensed Chunk Context, where document history is progressively propagated through compact memory states rather than unrestricted raw-token interactions. The resulting representations allow hundreds of long-document candidates to be jointly compared within a shared ranking context. Because document encoding is query-independent, the condensed representations can also be precomputed and cached offline, moving most long-document processing outside the online serving path.

To study this regime under realistic retrieval conditions, we further introduce LoDoc, a production-derived long-document reranking benchmark with broad candidate pools and substantially longer documents than conventional passage-level benchmarks. LoDoc contains 14,884 queries, with 206.9 candidates per query and 2,081.7 tokens per document on average. On the human-verified LoDoc test set, WeRank achieves 0.486 nDCG@10 and remains effective with up to 300 candidates. Controlled ablations further show that, under a matched representation budget, the proposed structured compression improves nDCG@10 from 0.400 with document-level compression to 0.443, a 10.8% relative gain, while additional experiments demonstrate favorable query-time efficiency and scalability.

In summary, our main contributions are as follows:

- We study a practically important yet insufficiently explored reranking regime involving hundreds of long-document candidates. To support research in this setting, we introduce *LoDoc*, a production-derived benchmark characterized by broad candidate pools, long documents, and diverse retrieval needs.
- We propose *WeRank*, a structured long-document compression framework for scalable listwise reranking. WeRank models documents at a fine-grained regional level with chunk-level document condensation and enables cross-region contextual communication by chaining condensed memory states.
- Extensive experiments demonstrate favorable ranking effectiveness, inference efficiency, and scalability to hundreds of long-document candidates, while controlled analyses validate the key design choices of WeRank. *Our dataset and code will be released.*

Table 1: **Dataset statistics**. Cand./Q denotes the average size of the query-specific candidate pool provided with the benchmark. For datasets without a predefined candidate pool, we report N/A. TREC DL19/20 (Craswell et al., 2020; 2021), two widely used benchmarks, are omitted because they share the MS MARCO corpus and thus exhibit the same document-length characteristics.

Dataset	#Query	#Corpus	Cand./Q	#Token/Doc	P25	P50	P75	P90
MS MARCO (Bajaj et al., 2016)	1.01M	8.84M	N/A	78.1	56	69	93	125
BEIR (Thakur et al., 2021)	53,359	33.9M	N/A	103.2	51	75	116	183
BRIGHT (SU et al., 2025)	1,384	1.14M	N/A	225.4	74	135	238	525
Our LoDoc	14,884	3.08M	**206.9**	**2,081.7**	**764**	**1,368**	**2,551**	**4,544**

2 LoDoc: A Long-Document Listwise Reranking Dataset

To study listwise reranking in the large-candidate, long-document regime, we construct *LoDoc*, a dataset derived from real-world production search traffic. LoDoc is designed to capture two characteristics central to our setting: broad candidate pools and long document content.

Dataset Collection and Annotation. We first sample approximately 20K queries from real-world search traffic. For each query, the production retrieval system retrieves up to 300 documents from a web-scale corpus, forming the candidate pool presented to the reranker.

We then apply an LLM-based filtering step (Yang et al., 2025) to remove ambiguous queries and cases with unclear search intent that are unsuitable for reliable ranking annotation. For the remaining queries, following the listwise annotation protocol of FullRank (Liu et al., 2025b), we employ Gemini 3.1 Pro to produce an ordered top-K list of document identifiers as ranking supervision. Queries for which the annotation API cannot provide a valid ranking, e.g., because of safety-sensitive candidate content, are further excluded.

To ensure reliable evaluation, the validation and test sets are fully reviewed by seven human experts. For each query, the experts jointly inspect the query, candidate documents, and API-generated top-K ranking, and revise the ordering when necessary. Examples are provided in A.2.2.

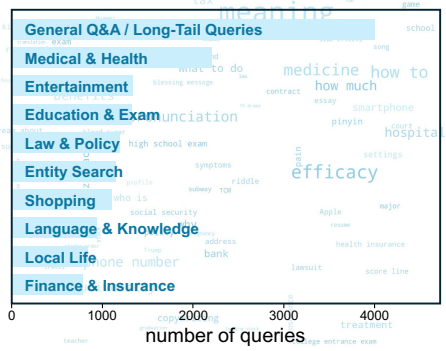


Figure 2: **Statistics of the LoDoc dataset**.

Dataset Characteristics. After filtering and annotation, LoDoc contains 14,884 queries and approximately 3.08M query-document candidate pairs, with 206.9 candidates per query on average. We split the dataset into 14,734 training, 50 validation, and 100 test queries, with ranking supervision defined as an ordered top-10 list. The candidate documents contain 2,081.7 tokens on average, with a median length of 1,368 tokens and a 75th-percentile length of 2,551 tokens. Together, the candidate-set size and document length correspond to approximately 431K candidate-document tokens per query on average, placing LoDoc naturally in the large-candidate, long-document regime considered in this work.

As shown in Table 1, the documents in LoDoc are substantially longer than those in commonly used reranking benchmarks such as MS MARCO, BEIR, and BRIGHT. Beyond scale, LoDoc also covers diverse practical search intents, whose distribution is illustrated in Fig. 2. This combination of broad candidate pools, long documents, and diverse information needs makes LoDoc suitable for studying large-scale long-document listwise reranking.

3 Preliminaries

3.1 Problem Formulation

Given a query q and a candidate set $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ returned by a retrieval system, the goal of reranking is to identify and reorder the K documents most relevant to q, where $K \ll N$. Each document $d_i(x_i^1, x_i^2, \ldots, x_i^{L_i})$ contains L_i textual tokens. We denote the ordered top-K ranking as $\pi_K = (\pi(1), \pi(2), \ldots, \pi(K))$, where π_K contains K distinct document indices selected from $\{1, \ldots, N\}$. The corresponding ranked documents are $\hat{\mathcal{D}}_K = (d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(K)})$. Our focus is the challenging regime in which both the candidate-set size N and document length L_i are large.

3.2 LISTWISE RERANKING AND ITS SCALING

A listwise reranker jointly conditions on the query and multiple candidates and directly predicts the ordered top-K ranking,

$$R_\theta(q, \mathcal{D}) \to \pi_K. \tag{1}$$

This formulation allows ranking decisions to depend on interactions among competing candidates, but its computational cost grows rapidly with both candidate-set size and document length. If the original document text is directly exposed to the reranker, the total input length can be written as $T_{\text{raw}} = |p| + |q| + \sum_{i=1}^{N} L_i \approx |p| + |q| + N\bar{L}$, where p denotes the ranking instruction and $\bar{L}$ is the average document length. Increasing either N or $\bar{L}$ enlarges the ranking context, while increasing both quickly makes direct listwise inference computationally expensive.

Existing methods mainly adopt two strategies to reduce this scaling burden. *Sliding-window reranking* (Liu et al., 2025a) partitions the candidate list into smaller subsets and ranks them through multiple local comparisons. This reduces the per-call context size, but restricts each invocation to only a portion of the candidate set and requires repeated processing across overlapping windows. On the other hand, *compression-based reranking* (Zhi et al., 2026) reduces the representation footprint of each candidate. Let a compressor C_ϕ map document d_i into a compact set of representations

$$Z_i = C_\phi(d_i) = \{z_i^1, z_i^2, \ldots, z_i^{M_i}\}, \qquad M_i \ll L_i. \tag{2}$$

The corresponding context length presented to the listwise reranker becomes

$$T_{\text{comp}} = |p| + |q| + \sum_{i=1}^{N} M_i, \tag{3}$$

which can be substantially smaller than T_{raw}. Compression is therefore particularly attractive for listwise reranking because it reduces the per-candidate context footprint while preserving the possibility of jointly comparing many candidates within a shared context.

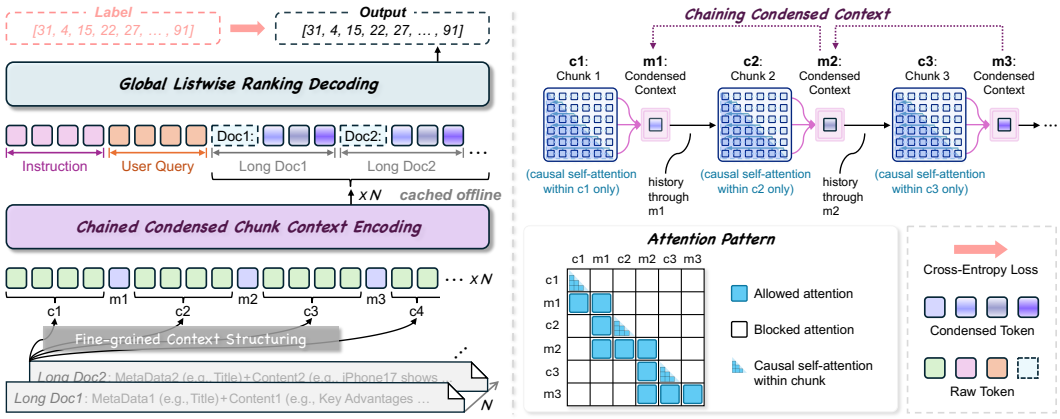
However, existing listwise rerankers, including compression-based methods, are mainly developed for short-passage settings. Extending them to the regime considered in this work is non-trivial because both the candidate-set size and document length are large. Long documents may contain ranking-relevant evidence distributed across multiple regions, while local content can depend on broader document context. As a result, compressing a long document into a compact representation is substantially more challenging than compressing a short passage. This leaves a central challenge: *how can long documents be compactly represented while modeling context across fine-grained document regions for large-scale listwise reranking?* We next introduce WeRank.

4 METHODOLOGY

4.1 OVERVIEW OF WERANK

To confront this challenge, WeRank centers on two key design considerations: how limited representation capacity should be allocated across a long document, and how contextual information should be modeled through the resulting compressed representations. WeRank addresses these through two complementary components: structured chunk-level document condensation and chained condensed chunk context modeling. The former explicitly distributes representation capacity across local document regions, while the latter enables cross-region contextual communication through compact memory states. WeRank adopts an encoder-decoder architecture in Fig. 3a. Given a candidate document, a query-independent encoder partitions the document into consecutive chunks and produces one condensed memory representation for each local region. These chunk-level representations are connected through a chained context pathway, so that information from preceding regions can progressively influence the representation of later chunks in Fig. 3b. For a candidate set of size N, the decoder jointly processes the query, ranking instructions together with the compact representations of all candidate documents and autoregressively generates the ordered top-K document identifiers.

This decomposition provides two practical advantages. First, the compact document representations reduce the context footprint exposed to the ranking decoder, allowing hundreds of long-document candidates to participate in the same listwise ranking context. Second, because document encoding is independent of the query, the condensed document representations can be precomputed and cached offline. Thus, online reranking only requires processing the query and cached candidate memories, moving most computation over the original long-document text outside the serving path.



(a) Overview of WeRank (b) Chained Condensed Chunk Context Encoding

Figure 3: **Detailed architectural illustration of the proposed WeRank framework.**

4.2 Chunk-Level Document Condensation

A central challenge in long-document compression is how to distribute a limited representation budget across the document. Existing document-level compression approaches form a fixed number of memory representations jointly over the entire input. While compact, such a formulation leaves the allocation of representation capacity across document regions implicit. This can become restrictive for long documents, where ranking-relevant information may be distributed across multiple local regions. We therefore adopt a fine-grained compression strategy that treats each chunk as an explicit compression unit and assigns it a dedicated memory position.

Given the i-th candidate document d_i, we partition its content into S_i consecutive chunks,

$$d_i = (c_i^1, c_i^2, \ldots, c_i^{S_i}), \tag{4}$$

where the s-th chunk is

$$c_i^s = (x_{i,s}^1, x_{i,s}^2, \ldots, x_{i,s}^{L_{i,s}}), \tag{5}$$

and $L_{i,s}$ denotes the number of textual tokens in the chunk.

For each chunk c_i^s, we append a dedicated memory token m_i^s, which provides an explicit compression position for that local region. After encoding, we retain the hidden state at the corresponding memory position, $h_i^s = H_i^s[\mathrm{pos}(m_i^s)]$, where H_i^s denotes the encoder hidden states for the s-th chunk. The resulting state h_i^s serves as the condensed representation of that document region. A lightweight adapter $\mathcal{A}_\psi$ maps each condensed state into the decoder input space, $z_i^s = \mathcal{A}_\psi(h_i^s)$. The final compact representation of document d_i is thus

$$Z_i = (z_i^1, z_i^2, \ldots, z_i^{S_i}), \tag{6}$$

containing one adapted condensed memory for each local document region.

This construction explicitly distributes the available representation capacity across the long document. Compared with reusing the hidden state of an ordinary content token, the dedicated memory position is explicitly introduced for information condensation. Compared with document-level memory compression under the same representation budget, chunk-level condensation structurally assigns each memory to a local document region rather than allowing all memories to be formed jointly over the entire document. As a result, WeRank retains a compact representation while providing finer-grained control over where information is preserved. Our ablation study in Sec. 5 validates the effectiveness of this chunk-level allocation strategy.

4.3 Chained Condensed Chunk Context

Chunk-level condensation determines where representation capacity is allocated, but by itself it does not specify how contextual information should propagate across the resulting local representations. Encoding each chunk independently preserves locality and maintains a compact compression interface, but fragments the document into isolated regions and prevents contextual information from crossing chunk boundaries. At the other extreme, allowing each chunk to directly attend to all

preceding raw tokens restores long-range context, but introduces an increasingly large contextual footprint and provides an uncompressed path that weakens the intended memory bottleneck.

We therefore introduce *Chained Condensed Chunk Context* modeling, which propagates document history through condensed memory states while keeping raw textual interaction local to each chunk.

For the first chunk, the encoder input is $U_i^1 = [c_i^1, m_i^1]$, while each subsequent chunk $s > 1$ is conditioned on the condensed representation of its immediately proceeding chunk,

$$U_i^s = [h_i^{s-1}, c_i^s, m_i^s]. \tag{7}$$

The encoder then produces

$$H_i^s = \mathrm{Enc}_\phi(U_i^s; \mathcal{M}_i^s), \qquad h_i^s = H_i^s[\mathrm{pos}(m_i^s)], \tag{8}$$

where $\mathcal{M}_i^s$ denotes the structured attention mask governing the chained context propagation, as illustrated in Fig. 3b.

Within each chunk, textual tokens follow standard causal self-attention. For $s > 1$, the preceding condensed state h_i^{s-1} is provided as a compact contextual prefix. Specifically, the j-th textual token in chunk s can access

$$\mathcal{V}(x_{i,s}^j) = \{h_i^{s-1}\} \cup \{x_{i,s}^1, \ldots, x_{i,s}^j\}, \tag{9}$$

while the memory position can attend to the preceding condensed state and the full current chunk,

$$\mathcal{V}(m_i^s) = \{h_i^{s-1}\} \cup c_i^s \cup \{m_i^s\}. \tag{10}$$

Importantly, later chunks cannot directly access raw tokens from earlier chunks, nor can they directly attend to all earlier condensed states. Instead, earlier document information can influence a later chunk only through the recursively propagated memory chain. The information flow is therefore $c_i^1 \to h_i^1 \to [h_i^1, c_i^2] \to h_i^2 \to \cdots \to [h_i^{S_i-1}, c_i^{S_i}] \to h_i^{S_i}$. This design creates a bounded interface to document history: regardless of how many earlier chunks exist, each new chunk receives only one compact historical state. Earlier information must therefore be progressively consolidated into the condensed memory before it can be propagated forward. This prevents historical context from bypassing the compression bottleneck while still allowing document-level information to accumulate across regions. In this sense, WeRank turns cross-chunk communication itself into a structured compression process, combining local fine-grained condensation and long-range contextual propagation without restoring unrestricted token-level interaction over the entire document. We compare alternative cross-chunk information-flow strategies in Sec. 5.

4.4 Global Listwise Decoding

After document encoding, each candidate d_i is represented by a compact sequence of adapted memories $Z_i = \{z_i^1, \ldots, z_i^{S_i}\}$. Given the ranking instruction p, query q, and N candidate documents, we construct the ranking decoder input as $X_{\mathrm{rank}} = [p, q, Z_1, Z_2, \ldots, Z_N]$. Each candidate is associated with a document identifier so that the decoder can explicitly refer to individual documents when generating the final ranking. The decoder jointly processes the complete candidate set and autoregressively outputs the ordered identifiers of the top-K documents.

Because each candidate is represented by only a small number of condensed memories rather than its original raw text, substantially more candidates can coexist within the same decoder context. This allows the ranking decision to be made through joint inter-document comparison rather than a sequence of locally overlapping ranking windows (Zhi et al., 2026).

A further practical advantage is that document encoding is query-independent. For each document d_i, its representation Z_i can therefore be computed once and cached prior to serving. During online reranking, WeRank only needs to process the query together with the cached candidate memories through the listwise decoder. As a result, the expensive processing of long raw documents is shifted largely offline, while joint candidate interaction is retained during online ranking. This benefit becomes increasingly important as either document length or candidate-set size grows.

4.5 Training WeRank

Training WeRank requires jointly learning fine-grained chunk condensation, cross-chunk context modeling, and large-scale listwise ranking. Directly optimizing all components at the target candidate scale is computationally expensive, since gradients must be maintained through the encoding

Table 2: **Quantitative comparison with existing listwise rerankers on the LoDoc test set**.

Method	nDCG@3	nDCG@5	nDCG@10	HRavg@3	HRavg@5	HRavg@10
PE-Rank (Liu et al., 2025a)	0.075	0.089	0.127	0.043	0.078	0.163
PE-Rank$_{100}$ (Liu et al., 2025a)	0.112	0.125	0.143	0.083	0.088	0.178
FullRank (Liu et al., 2025b)	0.334	0.319	0.306	0.237	0.218	0.242
C2R (Zhi et al., 2026)	0.197	0.209	0.219	0.143	0.176	0.220
Our WeRank	**0.515**	**0.504**	**0.486**	**0.390**	**0.392**	**0.411**

of many long documents. We therefore adopt a staged training strategy that progressively increases both representation complexity and candidate-set size. Concretely, we first perform a compression-specific warmup under the chunk-only setting, where only the newly introduced memory embeddings and adapter are optimized. We then conduct chunk-only SFT with 40 candidates to learn fine-grained document condensation, followed by full-model SFT with the chained condensed chunk context modeling scheme enabled to communicate cross-chunk information. Finally, we scale listwise training to larger candidate sets, e.g., 100 candidates, while freezing the encoder and optimizing the adapter and decoder. This staged procedure allows WeRank to first establish compact document representations at a tractable scale before adapting the ranking components to larger candidate sets.

5 Experiments

5.1 Experimental Setup

In this work, we report Hit Rate (HRavg@k) and normalized Discounted Cumulative Gain (nDCG@k) at $k \in \{3, 5, 10\}$. The two metrics capture complementary aspects of ranking quality. HRavg@k measures the overlap between the predicted and ground-truth top-k documents, while nDCG@k additionally accounts for their relative ordering by assigning larger gains to documents ranked higher in the ground-truth list and discounting them according to their predicted positions. Unless otherwise specified, following C2R (Zhi et al., 2026), our main evaluation focuses on reranking 100 candidates to produce a top-10 list. We additionally assess the scalability of WeRank with candidate sets of up to 300 documents. Unless otherwise specified, both the encoder and decoder are instantiated with Qwen3-1.7B. We set the maximum document length to 3,200 tokens and partition each document using a chunk size of 320 tokens. More details are provided in A.3.

5.2 Main Results on LoDoc

Ranking Effectiveness. Table 2 compares WeRank with three representative LLM-based listwise rerankers on LoDoc, including PE-Rank, FullRank, and C2R. For PE-Rank and C2R, we follow their sliding-window inference settings with a window size of 20 and a step size of 10. We additionally report PE-Rank$_{100}$, where all 100 candidates are placed within a single ranking context. PE-Rank and FullRank are retrained using their official implementations, while C2R is reimplemented following the original paper with the same backbone as WeRank. Since these baselines were originally developed for short-passage reranking, all methods share the same LoDoc evaluation protocol and candidate setting, but their original training and design regimes are not fully matched. As observed, WeRank achieves the best performance across all reported metrics, reaching 0.486 nDCG@10 and 0.411 HR$_{\text{avg}}$@10. FullRank obtains 0.306 nDCG@10 and 0.242 HRavg@10, while C2R reaches 0.219 and 0.220, respectively. PE-Rank100 also improves only marginally over the default sliding-window PE-Rank configuration, indicating that simply enlarging the shared ranking context is not sufficient in the long-document setting. These results demonstrate the effectiveness of WeRank for listwise reranking in the large-candidate, long-document regime.

Inference Efficiency. We next evaluate inference efficiency under the same 100-candidate setting. Table 3 separates online query-time ranking cost from offline query-independent document encoding cost. Compared with C2R, which uses the same backbone but requires nine overlapping ranking windows, WeRank performs the complete ranking in a single decoder invocation. This reduces the number of processed tokens from 2,988 to 1,529 (48.8%), generated tokens from 417 to 29 (93.0%), latency from 9.192s to 0.631s (93.1%), and online computation from 12.02 to 6.17 TFLOPs (48.7%). The comparison with FullRank further shows that single-pass inference alone does not guarantee efficiency. Although FullRank also requires only one model invocation, directly processing substantially longer textual inputs results in 24,717 prefill tokens and 669.27 TFLOPs, compared with 1,529 tokens and 6.17 TFLOPs for WeRank. WeRank does incur additional offline

Table 3: **Inference efficiency comparison**. Online cost includes the number of model invocations (#Calls), prefill tokens (#Proc.), generated tokens (#Gen.), latency, and computation. For methods with query-independent document encoding, offline cost is reported separately per document.

		Online Query-Time Cost					Offline Document Encoding	
Method	**nDCG@10**	**#Calls**	**#Proc.**	**#Gen.**	**Lat. (s)**	**TFLOPs**	**Lat./Doc (s)**	**TFLOPs/Doc**
PE-Rank	0.127	9	2,374	117	3.601	37.47	0.073	1.24
FullRank	0.306	1	24,717	41	7.805	669.27	N/A	N/A
C2R	0.219	9	2,988	417	9.192	12.02	0.068	1.65
WeRank	0.486	1	1,529	29	0.631	6.17	0.165	9.71

Table 4: **Ranking performance of WeRank under different candidate-set sizes**.

Candidate Budget	**nDCG@3**	**nDCG@5**	**nDCG@10**	**HRavg@3**	**HRavg@5**	**HRavg@10**
40 documents	0.570	0.598	0.607	0.435	0.473	0.578
100 documents	0.515	0.504	0.486	0.390	0.392	0.411
Up to 300 documents	0.489	0.470	0.437	0.375	0.356	0.346

document-encoding cost due to its structured long-document compression, but this computation is query-independent and can be performed once and reused across queries. Overall, WeRank shifts expensive long-document processing offline while keeping online listwise ranking very efficient.

Scaling with Candidate-Set Size. Finally, we evaluate WeRank as the candidate pool grows from 40 to 100 and up to 300 documents. As shown in Table 4, ranking naturally becomes more difficult as more candidates compete for the same top-10 positions. Nevertheless, WeRank continues to jointly process all candidate representations within a single ranking context and remains effective at substantially larger candidate scales. When the candidate budget is tripled from 100 to up to 300 documents, nDCG@10 decreases from 0.486 to 0.437, an absolute drop of only 0.049. These results demonstrate that the compact representations produced by WeRank support listwise reranking beyond the conventional tens-of-candidates setting, extending to hundreds of long-document candidates without changing the underlying inference paradigm.

5.3 Understanding WeRank

In this section, we conduct controlled analyses to understand the key design choices of WeRank.

Chunk-Level Document Condensation. We first examine the design of document condensation using variants (a)–(c) in Table 6. All variants produce the same number of document representations, thereby controlling the overall representation budget. Replacing ordinary last-token states with dedicated memory positions improves nDCG@10 from 0.394 to 0.415, demonstrating the benefit of explicitly introducing positions for information condensation. More importantly, under the same memory budget, assigning each memory to a local document chunk improves over jointly forming all memories at the document level, from 0.400 to 0.415. These results validate the chunk-level condensation design for preserving fine-grained information in long documents.

Chained Condensed Chunk Context. We next study how contextual information should propagate across chunk-level representations using variants (c)–(f) in Table 6. Starting from independent chunk compression (c), allowing unrestricted causal attention to preceding raw tokens (d) decreases nDCG@10 from 0.415 to 0.407. This suggests that simply introducing cross-chunk interaction is insufficient: direct access to historical raw tokens creates an uncompressed information path that weakens the intended compression bottleneck. Restricting cross-chunk communication to condensed memories (e) instead improves nDCG@10 to 0.420, showing that compressed historical states provide useful document-level context. However, each memory can still directly attend to all preceding memory tokens, allowing earlier information to bypass recursive consolidation into a single compact state. Our chained design (f) instead exposes each chunk only to the immediately preceding condensed memory, requiring historical information to be progressively compressed before being propagated forward. This further improves nDCG@10 to 0.443. These results demonstrate the benefit of recursively consolidating cross-chunk context through a compact memory chain.

Analysis of Condensed Memories. We further analyze whether condensed memories progressively accumulate useful document history. For each document, we retain only one memory from the Front,

Table 6: **Ablation study of compression schemes**, under a unified setup without staged training.

Index	Ablation Variants	nDCG@3	nDCG@5	nDCG@10
(a)	Document-Level Compression	0.411	0.425	0.400
(b)	Last-Token Chunk State	0.390	0.409	0.394
(c)	Chunk Compression Only	0.418	0.441	0.415
(d)	Chunk Compression with Full Causal Attention	0.409	0.419	0.407
(e)	Chunk Compression with Memory-only Causal Attention	0.427	0.442	0.420
(f)	Chained Chunk Context Compression	0.447	0.453	0.443

Middle, or Last position for ranking, while keeping the accompanying metadata unchanged. As shown in Fig. 4, Chunk Compression Only does not improve toward later positions, indicating that later document regions are not inherently more informative. Full Causal Attention also degrades from Front to Last despite having access to more preceding raw content, suggesting that greater historical access does not necessarily yield better compression.

In contrast, Chained Condensed Chunk Context improves progressively from Front to Middle to Last, with the largest advantage at the final memory. Since information from earlier regions must be recursively compressed before reaching later chunks, this trend indicates that chained propagation progressively integrates useful document history into later memory states. Together with Table 6, these results suggest that the benefit comes not simply from accessing more context, but from how historical information is progressively consolidated through compact memories.

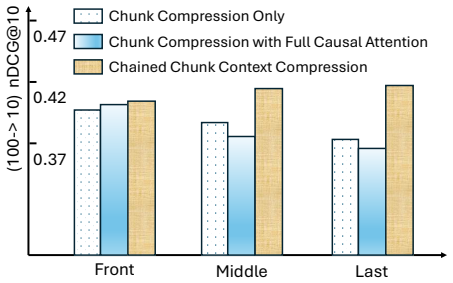


Figure 4: **Representation analysis**.

Effect of Training Strategy. Table 5 evaluates the staged training strategy introduced in Sec. 4.5. The baseline first performs full-model SFT with Chained Condensed Chunk Context on 40 candidates, and then scales training to 100 candidates by freezing the encoder and optimizing the adapter and decoder. This candidate-size curriculum achieves 0.443 nDCG@10. Adding a chunk-only warmup before chained training improves nDCG@10 to 0.457, suggesting that learning localized document condensation before introducing cross-chunk propagation provides a better initialization. Further prepending the compression-specific warmup, which adapts the newly introduced memory embeddings and adapter, increases nDCG@10 to 0.486. These results show that progressively introducing compression and cross-chunk modeling facilitates optimization at larger candidate scales.

Table 5: **Ablation study of the staged training strategy**.

Ablated Settings	nDCG@3	nDCG@5	nDCG@10
(a) Candidate-size Curriculum SFT	0.447	0.453	0.443
(b) + Chunk-only Warmup	0.475	0.474	0.457
(c) + Compression-specific Warmup	0.515	0.504	0.486

Effect of Model Scale. We further evaluate WeRank with Qwen3 backbones of different sizes. As shown in Table 7, performance improves consistently as model capacity increases, with nDCG@10 rising from 0.458 with the 0.6B model to 0.486 with 1.7B and 0.499 with 4B. This consistent trend shows that WeRank can benefit from increased backbone capacity.

Table 7: **Effect of backbone size on WeRank**.

Model Size	nDCG@3	nDCG@5	nDCG@10
WeRank-0.6B	0.478	0.471	0.458
WeRank-1.7B	0.515	0.504	0.486
WeRank-4B	0.537	0.521	0.499

6 Conclusion

In this work, we study listwise reranking in the challenging regime of hundreds of long-document candidates. To support research in this setting, we introduce LoDoc, a real-world benchmark characterized by broad candidate pools and long documents. We further propose WeRank, a scalable listwise reranking framework built on structured long-document compression. WeRank performs chunk-level document condensation and propagates cross-region context through chained condensed states, enabling compact global listwise comparison over large candidate sets. Extensive experiments demonstrate favorable ranking effectiveness, query-time efficiency, and scalability to hundreds of long documents, while controlled analyses validate the key design choices of WeRank.

AI USE STATEMENT

In this work, we used generative AI models through APIs as part of the annotation pipeline for the LoDoc dataset, including assisting with relevance assessment and listwise ranking of candidate documents. The resulting annotations were subject to human verification, and all validation and test instances were further reviewed and corrected when necessary by human experts. We did not use generative AI tools to rewrite the manuscript, conduct literature searches, or generate references. Generative AI was used occasionally for language polishing and improving the clarity of selected passages. All AI-assisted text was reviewed and edited by the authors. We take full responsibility for the final content of this work, including the data, analyses, claims, and text produced with the aid of generative AI.

ETHICS STATEMENT

The proposed WeRank framework is intended to support research on listwise reranking and does not introduce methods or content that are inherently harmful. The introduced LoDoc dataset is derived from real-world production search traffic, and we therefore take data privacy and responsible release seriously. The released dataset will include only data that can be redistributed under applicable data-governance and licensing requirements, with personally identifiable or sensitive information removed or filtered where necessary. We will also document the data provenance, annotation procedure, and relevant usage constraints to facilitate responsible use for research purposes only.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of this work. The architecture, attention design, and staged training procedure of WeRank are described in the main paper. The construction, annotation, and data-processing pipeline of LoDoc are also documented in detail. Experimental settings, evaluation protocols, hyperparameters, and additional implementation details are provided in the Experiments section and Appendix. Upon publication, we will publicly release the source code and LoDoc dataset, together with model configurations, data-processing scripts, and easy-to-follow instructions for reproducing the main results.

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.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

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

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*, 2020.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the trec 2020 deep learning track. *arXiv preprint arXiv:2102.07662*, 2021.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024.

Jyun-Yu Jiang, Chenyan Xiong, Chia-Jung Lee, and Wei Wang. Long document ranking with query-directed sparse transformer. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4594–4605, 2020.

Fanfan Liu and Haibo Qiu. Context cascade compression: Exploring the upper limits of text compression. *arXiv preprint arXiv:2511.15244*, 2025.

Qi Liu, Bo Wang, Nan Wang, and Jiaxin Mao. Leveraging passage embeddings for efficient listwise reranking with large language models. In *Proceedings of the ACM on Web Conference*, pp. 4274–4283, 2025a.

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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 162–176, 2025b.

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.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

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

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 708–718, 2020.

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

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

Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. First: Faster improved listwise reranking with single token decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8642–8652, 2024.

Hongjin SU, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han yu Wang, Liu Haisu, Quan Shi, Zachary S Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan O Arik, Danqi Chen, and Tao Yu. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *The Thirteenth International Conference on Learning Representations*, 2025.

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. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 14918–14937, 2023.

Yiqun Sun, Pengfei Wei, and Lawrence B. Hsieh. Very efficient listwise multimodal reranking for long documents. In *Forty-third International Conference on Machine Learning*, 2026.

Jiakai Tang, Sunhao Dai, Kun Wang, Zhiluohan Guo, Yu Zhao, Cong Fu, Kangle Wu, Yabo Ni, Anxiang Zeng, Xu Chen, et al. Onerank: Unified transformer-native ranking architecture for multi-task recommendation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 8053–8064, 2026.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhewei Zhi, Yingyi Zhang, Yizhen Jing, Xianneng Li, Jianing Liu, Huajie Liu, and Yongliang Ding. Compress-then-rank: Faster and better listwise reranking with large language models via ranking-aware passage compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35059–35067, 2026.

Changjiang Zhou, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Fan Yixing, and Xueqi Cheng. Longranker: Efficient one-pass document reranking with long-context large language models. In *Proceedings of the ACM Web Conference 2026*, pp. 2004–2013, 2026.

Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, et al. Rankmixer: Scaling up ranking models in industrial recommenders. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6309–6316, 2025.

A Appendix

A.1 Related Work

Ranking in Retrieval Systems. Modern search and recommendation systems typically adopt multi-stage pipelines, where a large candidate pool is progressively reduced through retrieval, ranking, and reranking stages. Recent industrial ranking models, such as RankMixer and OneRank, further explore scalable Transformer-based architectures under practical serving constraints (Zhu et al., 2025; Tang et al., 2026). For text reranking, pointwise and listwise ranking represent two widely adopted formulations. Pointwise rerankers independently estimate the relevance of each query-document pair, as exemplified by MonoBERT, MonoT5, and RankLLaMA (Nogueira et al., 2019; 2020; Ma et al., 2024). Such independent scoring is naturally parallelizable and well suited to latency-sensitive settings, but does not explicitly model the relative relationships among competing candidates. In contrast, listwise reranking (Sun et al., 2023) jointly considers multiple candidates under a shared query context, allowing relevance judgments to be made comparatively. This formulation is particularly appealing for reranking, where the objective is ultimately to identify the most relevant items from a competitive candidate set.

LLM-based Listwise Reranking. Thanks to the powerful instruction-following and contextual reasoning capabilities of large language models, LLMs have enabled listwise rerankers that directly reason over multiple candidates and generate their relative ordering. RankGPT prompts an LLM to generate a permutation of candidate identifiers and uses sliding-window inference when the full candidate list exceeds the context window (Sun et al., 2023). RankZephyr distills such listwise ranking behavior into an open-source reranker, while FIRST improves decoding efficiency by deriving the ranking from candidate-identifier logits at the first decoding step (Pradeep et al., 2023b; Reddy et al., 2024). FullRank moves beyond sliding windows by placing the full candidate list into a single context for one-pass listwise ranking, while limiting individual candidate lengths to fit within the overall context budget (Liu et al., 2025b). LongRanker further improves reranking through hierarchical intra- and inter-document positional modeling together with attention pruning (Zhou et al., 2026). These advances progressively reduce repeated inference and strengthen global candidate interaction. However, much of this line of work focuses on accommodating candidate texts within the LLM context and is primarily evaluated in passage-oriented or relatively short-document settings. Consequently, directly extending full-text listwise inference to hundreds of long documents remains computationally demanding.

Latent Memory and Context Compression. Beyond the reranking literature, a parallel line of research on long-context LLMs replaces lengthy textual inputs with compact latent representations to extend effective context length and reduce inference cost. RMT (Bulatov et al., 2022) propagates recurrent memory tokens across successive segments, while AutoCompressors (Chevalier et al., 2023) summarize segmented contexts into reusable soft memories and accumulate previous summaries for subsequent processing. Other approaches, including Gisting (Mu et al., 2023), ICAE (Ge et al., 2024), and C3 (Liu & Qiu, 2025), learn compact latent representations for prompt compression, context reconstruction, or efficient long-context modeling. These studies establish the feasibility of replacing raw textual context with learned memory representations, and AutoCompressors further explore precomputed representations for passage reranking. WeRank builds on this general principle but adapts it to long-document listwise ranking. Rather than compressing a document into a globally formed set of memories, WeRank distributes representation capacity across local document regions and propagates historical information through a bounded chain of condensed states. This propagation pathway is related in spirit to segment-level memory mechanism such as RMT, but WeRank retains the condensed states across document regions as the cached document representation and optimizes them directly through the downstream listwise ranking objective. Our ablations further examine these design choices against document-level compression, accumulated-memory interaction, and unrestricted cross-chunk attention.

Compressed Representations for Listwise Reranking. More directly related to WeRank, recent rerankers compress candidate texts before listwise reasoning. PE-Rank compresses each passage into a single embedding token and performs listwise ranking over the resulting compact representations (Liu et al., 2025a). C2R extends this paradigm to ranking-aware multi-vector passage compression, jointly optimizing the compressor and reranker for listwise ranking (Zhi et al., 2026). Despite this progress, these approaches are primarily developed and trained in short-passage settings

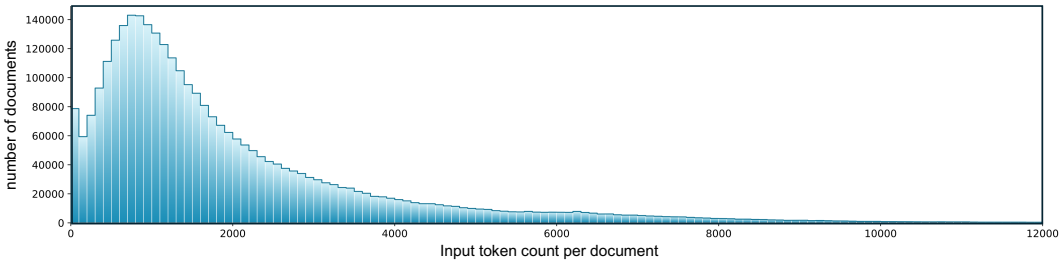


Figure 5: **Distribution of document lengths across the LoDoc dataset**, where token counts are computed on raw, untruncated document content.

and do not directly target the joint regime of *long individual documents* and *hundreds of candidates*. WeRank is explicitly designed for this regime by distributing representation capacity across local document regions and controlling how contextual information propagates through the resulting condensed states. This structured representation enables hundreds of long-document candidates to participate jointly in global listwise reasoning.

A.2 More Details of LoDoc Datset

A.2.1 Additional Statistics

We provide additional statistics of LoDoc to characterize the long-document, large-candidate regime considered in this work. Fig. 5 shows the distribution of raw, untruncated document lengths over 3,080,062 documents. The distribution is right-skewed, with a substantial portion of documents containing hundreds to thousands of tokens and a long tail extending beyond 10,000 tokens. This indicates that long documents are not limited to a small number of outliers in LoDoc. Accordingly, short fixed-length passage truncation may discard substantial portions of the raw document content.

Fig. 6 further reports the distribution of candidate-set sizes across queries. Candidate pools vary considerably across queries, with many queries associated with hundreds of documents. The distribution also shows a concentration toward the upper end of the candidate range, which is consistent with the maximum candidate size used by the production retrieval pipeline. Overall, the distribution motivates our evaluation of listwise reranking under large candidate-set sizes, including settings with hundreds of documents.

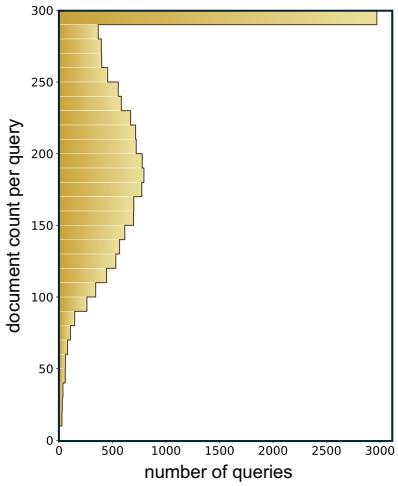


Figure 6: **Distribution of candidate-set sizes across the LoDoc dataset**.

A.2.2 Data Examples

We provide several representative ranking cases from the LoDoc dataset to illustrate the input format and supervision in Table 9, Table 10 and Table 11. Each case consists of the ranking instruction, a user query, example candidate long documents, the corresponding WeRank input format, and a ground-truth top-10 ranking. To improve readability, only two candidate documents are shown for each case and their contents are truncated. LoDoc primarily contains English and Chinese, with a small proportion of other languages. For presentation, non-English text is translated into English.

A.2.3 Additional Details

Candidate Construction. For each query, we first obtain a source candidate pool from the production retrieval system, containing up to 300 documents. As described in the dataset construction pipeline, high-scoring candidates are further annotated to obtain an ordered ground-truth top-10 list. The resulting source pool therefore always contains all documents in the ground-truth top-10. To construct the candidate sets used in our 40- and 100-candidate settings, we always retain these 10

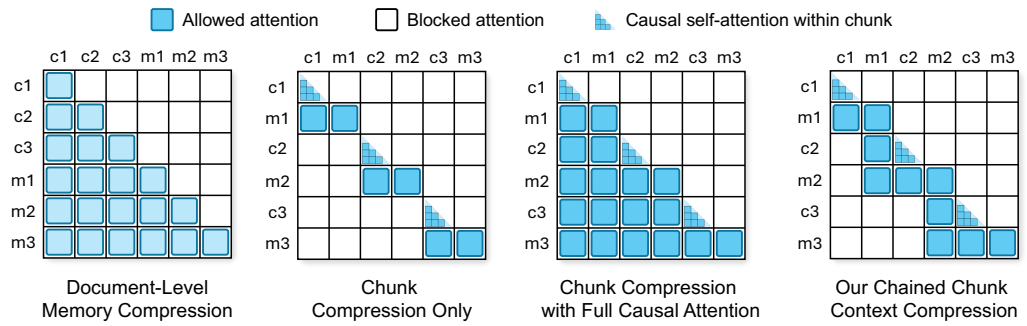


Figure 7: **Attention patterns of representative compression variants**. Blue cells denote allowed attention, white cells denote blocked attention, and triangular regions indicate causal self-attention within each content chunk.

ground-truth documents and randomly sample additional candidates from the remaining source pool up to the corresponding candidate budget. If fewer candidates are available than the specified budget, all available candidates are retained. This construction preserves the target top-10 ranking while naturally allowing the actual candidate-set size to vary across queries.

Candidate Randomization and Ranking Targets. We do not preserve the original retrieval order when presenting candidates to the reranker. Instead, the candidates are randomly permuted during training, with a new permutation sampled at each epoch to reduce potential position bias. Document identifiers are assigned according to the permuted candidate order, and the target top-10 identifier sequence is remapped accordingly while preserving the ground-truth ranking order.

A.2.4 Sensitive Information

The LoDoc dataset does not include private messages, user profiles, browsing histories, platform-internal records, or other non-public user information. We further review and filter the collected documents to remove personally identifiable information, sensitive content, and inappropriate or offensive content where necessary. Only data that can be redistributed under applicable licensing requirements are included in the released version. Therefore, the released dataset will be restricted to public-facing content and does not contain non-public personal or user-level information.

A.3 More Experimental Details

More Implementation Details. By default, WeRank employs Qwen3-1.7B (Yang et al., 2025) for both the chunk-level encoder and the ranking decoder. A lightweight adapter, consisting of a LayerNorm followed by a two-layer FFN with a hidden dimension of 4,096, bridges the condensed chunk representations to the decoder and introduces approximately 17M additional parameters. Following the staged training strategy described in Sec. 4.5, training proceeds first with 40 candidates and is subsequently scaled to 100. We begin with a *compression-specific warmup* under the chunk-only setting, optimizing only the newly introduced memory-token embedding and adapter while keeping the pretrained encoder and decoder frozen (2 epochs, $\mathrm{lr} = 1 \times 10^{-4}$). We then perform *chunk-only SFT*. The encoder and compression-specific parameters are first optimized with the decoder frozen (3 epochs, $\mathrm{lr} = 3 \times 10^{-5}$), after which the decoder is unfrozen for full-model optimization for another 3 epochs ($\mathrm{lr} = 1 \times 10^{-5}$). Starting from this checkpoint, we enable chained condensed context and perform *chained-context SFT*, again with 40 candidates. The entire model remains trainable, using a conservative learning rate to avoid substantially perturbing the previously learned document representations (3 epochs, $\mathrm{lr} = 1 \times 10^{-5}$). Finally, we scale listwise training from 40 to 100 candidates. Because end-to-end encoding of 100 long-document candidates is memory-intensive, the encoder is frozen and only the adapter and ranking decoder are optimized (3 epochs, $\mathrm{lr} = 1 \times 10^{-6}$). All stages use AdamW and a cosine learning-rate schedule on 64 NVIDIA H20 GPUs. The per-device batch size is 2 with a gradient accumulation factor of 2, resulting in an effective batch size of 256. Unless otherwise specified, we set the maximum document input length to 3,200 content tokens and partition each document into chunks of 320 tokens. Training uses candidate-set sizes of 40 and 100, while the scaling experiments additionally evaluate candidate sets of up to 300 documents.

Document Length and Chunk Size. The maximum document length and chunk size are chosen by jointly considering the empirical document-length distribution and the practical storage budget for cached document representations. As shown in the LoDoc statistics, document lengths exhibit a pronounced long-tail distribution, with the 85th percentile at approximately 3,200 tokens. We thus set the maximum document input length to 3,200 content tokens, while bounding the computation required for the long tail. The chunk size is further determined by the storage budget for cached representations. In the production retrieval setting, the cached content representation of each document is allocated a storage budget of approximately 80 KB. Each condensed memory is a 2,048-dimensional vector, and storing 10 such vectors in FP32 requires approximately $2{,}048 \times 10 \times 4 \approx 80$ KB per document. We therefore represent the document content with at most 10 chunk-level memories. Given a maximum input length of 3,200 tokens and a budget of at most 10 memories, we set the chunk size to 320 tokens. This choice provides a practical balance between fine-grained document condensation and the storage overhead of cached representations.

Baseline Coverage. We discuss LongRanker (Zhou et al., 2026) in the related work but do not reimplement it in the numerical comparison because its original implementation relies on a different candidate-ordering protocol. Concretely, its top-k attention pruning mechanism operates on the leading candidates returned by the first-stage retriever, whereas our evaluation randomizes the candidate order. Adapting LongRanker to this setting would therefore require modifying its attention-selection protocol, making the resulting comparison less faithful to the original method.

Attention Patterns of Compression Variants. Fig. 7 visualizes the attention patterns of representative compression variants evaluated in the ablation study. The *Document-Level Memory Compression* baseline allows all memory tokens to aggregate information from the preceding document content without explicit chunk-wise allocation. *Chunk Compression Only* instead assigns a dedicated memory to each chunk, while preventing information exchange across chunks. *Chunk Compression with Full Causal Attention* introduces cross-chunk interaction by allowing later chunks and memories to directly attend to all preceding raw content and memories. In contrast, our *Chained Chunk Context Compression* restricts historical information propagation to the condensed memory from the preceding chunk, thereby maintaining a bounded information pathway across the document. These patterns provide a structural view of the variants compared in Table 6.

Statistical Significance Analysis. To assess the reliability of the observed improvements, we conduct paired statistical analyses on the test set of LoDoc. For each method, we compute the performance difference in nDCG@10 against our WeRank. We estimate 95% confidence intervals using 10,000 paired bootstrap resamples and assess statistical significance using a two-sided paired randomization test with 100,000 permutations. As shown in Table 8, the paired-bootstrap confidence intervals remain strictly above zero, and the paired randomization tests yield <0.001. These results provide consistent statistical support for the improvements observed on the LoDoc test set.

Table 8: **Paired statistical analysis of WeRank against external baselines on the LoDoc test set**. Δ denotes the mean performance difference in nDCG@10. The 95% confidence intervals are estimated using 10,000 paired bootstrap resamples, and p-values are obtained from two-sided paired randomization tests with 100,000 permutations.

WeRank versus	Δ	95% CI	p
vs. PE-Rank	+0.359	[+0.3128, +0.4094]	<0.001
vs. PE-Rank$_{100}$	+0.343	[+0.2944, +0.3924]	<0.001
vs. FullRank	+0.180	[+0.1242, +0.2376]	<0.001
vs. C2R	+0.267	[+0.1910, +0.3043]	<0.001

A.4 Discussion and Outlook

WeRank provides an initial step toward global listwise reranking over hundreds of long documents, but several directions remain open. First, the current condensed representations are learned primarily through ranking supervision. Large-scale compression-oriented pretraining, with objectives such as document question answering or continuation, may further improve the ability of compact memories to preserve document semantics and ranking-relevant information. Second, WeRank currently relies on supervised fine-tuning for listwise ranking; ranking-oriented reinforcement learning may further improve ranking quality and output consistency. Beyond these modeling directions, our current study focuses on textual long documents. Extending structured compression to multimodal long documents could enable joint reasoning over textual and visual content (Sun et al., 2026). Finally, although the validation and test sets are carefully human-verified, expanding the scale of human-verified evaluation would provide a broader assessment of large-scale long-document reranking.

Table 9: **Case 1 on the LoDoc dataset.** A representative example illustrating the input format and ranking supervision of WeRank, including a user query, candidate long documents, and the corresponding ground-truth top-10 ranking. For readability, only two example candidate documents are shown, and their contents are truncated.

Ranking Instruction

You are an expert in information retrieval. Given a search query and a set of candidate documents, your task is to comprehensively compare the documents based on their relevance to the query, how well they satisfy the user's information needs, and the overall quality of their content.

Select the top 10 documents that best satisfy the query and rank them in descending order of overall relevance and quality.

Output format: Output exactly 10 document IDs, ordered from the most relevant and satisfying to the least, separated by commas. Do not output any other content.

For example: [17, 121, 235, 53, 118, 5, 79, 20, 1, 39]

The search query and candidate documents are provided below:

User Query

What causes arm numbness after waking up?

Candidate Documents (Raw Text)

Document 1

Title: Shoulder and Arm Pain with Hand Numbness: Beware of Thoracic Outlet Syndrome!
Publish Time: December 2, 2024

Content: Shoulder and arm pain, hand numbness, and weakness—what could be causing these symptoms? Many people attribute them to neck and shoulder muscle fatigue after long hours of work and assume that rest will help. However, one condition that should not be overlooked is thoracic outlet syndrome.
1. What is thoracic outlet syndrome? Thoracic outlet syndrome refers to a group of neurological and vascular symptoms in the upper extremities caused by compression of the brachial plexus, subclavian artery, or subclavian vein at the thoracic outlet. Neurogenic thoracic outlet syndrome is the most common type, accounting for approximately 90%-95% of cases. Common symptoms include upper-limb weakness, numbness, abnormal sensations, and non-radicular pain. Typical neurogenic thoracic outlet syndrome also present with atrophy of the intrinsic hand muscles and sensory abnormalities along the medial forearm and ulnar side of the hand.
2. Where is the thoracic outlet? The thoracic outlet is the region between the clavicle and the first rib, extending from the supraclavicular fossa to the axilla. It contains ...

Document 2

Title: Warning: Numb Hands After Waking Up May Be a Sign from Your Body!
Publish Time: March 14, 2026

Content: After waking up, you may suddenly notice that your hand feels numb. What causes hand numbness during sleep? Could it be related to cervical spondylosis, and how can it be prevented? Arm numbness during sleep may be associated with impaired blood circulation. An improper sleeping posture or keeping the arm in the same position for a long time can restrict circulation and cause numbness. This may occur more frequently in summer when people sleep with air conditioning.
Although many people do not pay much attention to occasional hand numbness, cervical spondylosis can also cause numbness in the arms, particularly in middle-aged and older adults. In addition to hand numbness, cervical spondylosis may be accompanied by soreness or pain in the neck and shoulders and discomfort in the upper limbs. Common causes include the following:
(1) Improper sleeping posture: Sleeping in an unsuitable position, such as placing the hand or arm underneath the body for an extended period, may impair circulation and cause numbness. (2) Pillow height: Using a pillow that is too high may also affect circulation and ...

... many additional candidate long documents ...

Actual Input to WeRank

Instruction: The ranking instruction shown above.
Query: `#{user query}`
Candidate Documents:

```
Doc 1:  #{compressed representation of document 1}
Doc 2:  #{compressed representation of document 2}
...
Doc N:  #{compressed representation of document N}
```

Ground Truth [56, 51, 2, 9, 207, 217, 234, 11, 73, 137]

Table 10: **Case 2 on the LoDoc dataset.** A representative example illustrating the input format and ranking supervision of WeRank, including a user query, candidate long documents, and the corresponding ground-truth top-10 ranking. For readability, only two example candidate documents are shown, and their contents are truncated.

Ranking Instruction

You are an expert in information retrieval. Given a search query and a set of candidate documents, your task is to comprehensively compare the documents based on their relevance to the query, how well they satisfy the user's information needs, and the overall quality of their content.

Select the top 10 documents that best satisfy the query and rank them in descending order of overall relevance and quality.

Output format: Output exactly 10 document IDs, ordered from the most relevant and satisfying to the least, separated by commas. Do not output any other content.

For example: [17, 121, 235, 53, 118, 5, 79, 20, 1, 39]

The search query and candidate documents are provided below:

User Query

Can I keep a cat in a basement?

Candidate Documents (Raw Text)

Document 1

Title: Don't Be Misled: You Can Still Keep a Cat in a Small Home, as Long as You Keep These Points in Mind

Publish Time: April 22, 2025

Content: Have you ever wanted to keep a cat but been discouraged by claims that a small home is unsuitable for cats?
Some people say that cats need a large space to run and jump, or that keeping a cat in a small apartment is unfair to the animal. In fact, what matters most is not the amount of floor space, but how effectively vertical space is used. With proper planning, even a small apartment can become a comfortable environment for a cat.
As someone who keeps a cat in a 35-square-meter rental apartment, here are several practical tips for cat owners living in small spaces.
1. Make full use of vertical space. For cats, climbing and moving vertically can be more attractive than simply running across the floor. Instead of trying to create more floor space, owners can make use of walls and elevated surfaces to expand the cat's activity area.
· Cat trees: Choose a tall and stable cat tree. It does not need to occupy much floor space, but should allow the cat to climb and observe its surroundings.
· Floor-to-ceiling climbing poles: These are especially useful in small homes because ...

Document 2

Title: Stop Harming Your Cat: These 6 Everyday Habits May Be More Harmful Than You Think

Publish Time: May 7, 2023

Content: Animal behavior experts warn that some seemingly harmless everyday habits may gradually harm cats. Over time, these behaviors can undermine a cat's trust, affect its health, and reduce its sense of security at home. The following six common habits may be more harmful than many owners realize.
1. Shouting at a cat when it misbehaves. When a cat causes trouble, owners may raise their voice in an attempt to correct its behavior. However, cats do not understand cause and effect in the same way humans do. Instead of learning that a specific action was wrong, the cat may simply perceive its owner as unpredictable or threatening. Repeated scolding can increase stress and may contribute to behaviors such as hiding, aggression, or inappropriate urination.
2. Never giving the cat opportunities to hunt or chase during play. Many indoor cats go for days without engaging in activities that resemble natural hunting behavior. Cats are predators by nature, and behaviors such as stalking, chasing, pouncing, and catching are deeply instinctive. If these behaviors cannot be expressed for long periods, ...

... many additional candidate long documents ...

Actual Input to WeRank

Instruction: The ranking instruction shown above.
Query: `#{user query}`
Candidate Documents:

```
Doc 1:  #{compressed representation of document 1}
Doc 2:  #{compressed representation of document 2}
...
Doc N:  #{compressed representation of document N}
```

Ground Truth [36, 157, 42, 49, 149, 41, 82, 100, 79, 18]

Table 11: **Case 3 on the LoDoc dataset.** A representative example illustrating the input format and ranking supervision of WeRank, including a user query, candidate long documents, and the corresponding ground-truth top-10 ranking. For readability, only two example candidate documents are shown, and their contents are truncated.

Ranking Instruction

You are an expert in information retrieval. Given a search query and a set of candidate documents, your task is to comprehensively compare the documents based on their relevance to the query, how well they satisfy the user's information needs, and the overall quality of their content.

Select the top 10 documents that best satisfy the query and rank them in descending order of overall relevance and quality.

Output format: Output exactly 10 document IDs, ordered from the most relevant and satisfying to the least, separated by commas. Do not output any other content.

For example: [17, 121, 235, 53, 118, 5, 79, 20, 1, 39]

The search query and candidate documents are provided below:

User Query

Differences between PET and PE materials

Candidate Documents (Raw Text)

Document 1

Title: How to Distinguish PET Plastic Bottles from PE Plastic Bottles

Publish Time: January 15, 2026

Content: 1. How to distinguish them in everyday use
· Transparency: PET plastic bottles are generally more transparent than PE bottles. PET typically has high transparency, while PE is relatively less transparent and may sometimes appear milky white.
· Bottle opening: The inner wall of a PET bottle opening is usually neat and smooth, with clearly defined threads. In contrast, PE bottle openings may be somewhat rougher, and the threads may be less distinct.
· Bottle bottom: PET bottles usually have a small dot on the bottom, whereas PE bottles may have a line or other markings.
· Recycling code: Plastic bottles usually have a triangular recycling symbol on the bottom containing a number. PET bottles are labeled with the number "1." PE bottles include HDPE (high-density polyethylene) and LDPE (low-density polyethylene), which are labeled "2" and "4," respectively. HDPE, labeled "2," is more commonly encountered in everyday use.
· Touch and sound: When squeezed by hand, PET bottles tend to produce . . .

Document 2

Title: Don't Misuse Plastic Products: Here's How to Identify Them

Publish Time: December 26, 2019

Content: Plastic products are everywhere in daily life, including cups, tableware, straws, takeout containers, and shopping bags. At supermarkets and markets, free plastic bags are often provided for weighing and carrying groceries. Many people also take extra bags home to separate vegetables and meat, or wrap food before placing it directly in the refrigerator.
However, this may not always be appropriate. Plastics include many different materials: some have poor heat resistance, while others are unsuitable for contact with oil. Using the wrong type of plastic may pose health risks.
1. Supermarket plastic bags are not suitable for hot food or all types of meat. Most plastic bags provided by regular supermarkets are made of PE. They are generally suitable for carrying everyday items at room temperature, but may not always be appropriate for storing fresh food.
Only food-grade PE plastic bags should be used for food storage. It is important to note that only PE bags specifically manufactured as food-grade products should be used to store fresh food. PE is a broad category of materials, and . . .

. . . many additional candidate long documents . . .

Actual Input to WeRank

Instruction: The ranking instruction shown above.
Query: `#{user query}`
Candidate Documents:

```
Doc 1:  #{compressed representation of document 1}
Doc 2:  #{compressed representation of document 2}
...
Doc N:  #{compressed representation of document N}
```

Ground Truth [24, 1, 69, 150, 55, 112, 85, 142, 74, 164]