Zero-Shot Reranking with Large Language Models and Precomputed Ranking Features: Opportunities and Limitations

Maria Movin*
Spotify AB
Stockholm, Sweden
mariamovin@spotify.com

Claudia Hauff
Spotify AB
Delft, Netherlands
claudiah@spotify.com

Abstract

LLMs have been explored for their use in IR as end-to-end rankers, rerankers and assessors. Recently, the exploration of the prompt-and-predict paradigm for reranking in combination with highly performant LLMs have drawn the attention of researchers. Instead of training or fine-tuning a reranker, LLMs are prompted in a zero-shot manner to produce relevance scores, pairwise preferences, or reranked lists. Existing research, though, has been confined to *unstructured* text corpora, leaving a gap in our understanding: to what extent do the findings of zero-shot LLM rerankers established on plain text corpora hold for datasets containing predominantly precomputed ranking features as is common in industrial settings? We explore this question via an empirical study on one public learning-to-rank dataset (MSLR-WEB10K) and two datasets collected from an audio streaming platform's search logs. Our results paint a differentiated picture: *On average*, there remains a significant performance gap: prompting the high-capacity LLM GPT-4 results in up to 16% lower NDCG@10 compared to the traditional supervised learning-to-rank (LTR) approach LambdaMART on the public MSLR-WEB10K dataset. However, when focusing *only* on a subset of hard queries—i.e. queries where the LTR approach ranks a non-relevant document at the top—the zero-shot LLM reranking outperforms the LTR baseline. We confirm the same trends on two proprietary audio search datasets. We also provide insights into prompt design choices and their impact on LLM reranking. We show that LLMs remain brittle, with the same strategies sometimes helping or hurting depending on the model size and dataset.

CCS Concepts

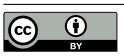
• **Information systems** → **Learning to rank**; **Language models**.

Keywords

Zero-Shot Ranking; Large Language Models; Learning to Rank

ACM Reference Format:
Maria Movin and Claudia Hauff. 2025. Zero-Shot Reranking with Large Language Models and Precomputed Ranking Features: Opportunities and Limitations. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25), July 13–18, 2025, Padua, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3726302.3730119

*Also with Department of Computer and Systems Sciences, Stockholm University.



SIGIR '25, Padua, Italy

ACM ISBN 979-8-4007-1592-1/2025/07
https://doi.org/10.1145/3726302.3730119

1 Introduction

Large language models (LLMs) have been explored for their use in information retrieval (IR) as possible rerankers [46, 54] (often starting with a simple lexical retrieval as first retrieval stage) and as assessors [16, 51]. While there is still a debate around their utility for the latter [15], their effectiveness as rerankers has been empirically established. Recently, the exploration of the prompt-and-predict paradigm [29] for (re)ranking in combination with highly performant LLMs such as GPT-4 and Flan-T5 have drawn the attention of researchers [39, 40, 42, 49, 54, 55]. These models are already instruction fine-tuned and can be considered *zero-shot* or *few-shot* (re)rankers. Thus, instead of fine-tuning a cross-encoder [8, 35, 38] or bi-encoder [18, 22, 24, 53] which requires a considerable amount of training data, we no longer require a training step and instead rely on the LLM as-is for (re)ranking.

Although the initial results of zero-shot LLM-based rerankers are promising, existing research has been largely limited to a few corpora, predominantly MS MARCO [1] and smaller corpora such as TREC-COVID [44] and NQ [25] available in the BEIR [50] benchmark. Importantly, these corpora contain very little *structured* data. Concretely, MS MARCO consists only of plain text passages, TREC-COVID comprises biomedical literature with some metadata and structure, and NQ includes questions and corresponding Wikipedia pages with minimal structure. In contrast, many real-world datasets, especially those in industrial settings, contain significant amounts of structured information or consist entirely of precomputed ranking features. In what is by now called "traditional" machine learning, the effective use of LTR models with hundreds or thousands of input features requires significant machine learning expertise. But what if it were possible to simply prompt an LLM with those features and get similarly good results? Formally, in our work, we explore **to what extent the findings of zero-shot LLM *rerankers* established on plain text corpora hold for datasets consisting of hundreds of precomputed ranking features**. This examination is crucial, as many ranking problems solved in practical applications leverage extracted features extensively (which in turn led to the establishment of the LTR subfield [31] in IR), and understanding how LLMs handle these features can provide insights into both their versatility and potential limitations. We focus our investigation on the top-k reranking task, which can be explained by the industrial setups we consider. In these setups, a multi-stage ranking cascade is typically employed, where successively more expensive rankers rerank the top-k ranking from the previous stage [27, 30, 52].

To answer our question, we conducted experiments on three datasets with precomputed ranking features: (i) the public LTR dataset MSLR-WEB10K [41], which includes 136 named features

(e.g. *body stream length*, *anchor min of tf*idf*)[1]; and (ii) two proprietary LTR datasets, Audio24 and Audio24-Hard, each containing 300 features derived from search logs of a large audio streaming platform. We note here that although a number of additional LTR datasets have been released over the years (e.g. the Istella22 [10] and Yahoo! Learning to Rank Challenge [4] datasets), they are typically distributed in a purely numerical form where each feature name is represented by a numeric ID without any semantic context. Such datasets are not suitable for our purposes[2].

We employ the high-capacity API-only accessible GPT-4 (gpt-4-turbo-2024-04-09) [37] and the open-source Llama-3.1 8B model (Llama-3.1-8B-Instruct) [17] in a zero-shot setting as rerankers. We choose a high capacity proprietary model and an open-source modestly-sized model as two ends of the spectrum that practitioners might be interested in applying commonly. We compare the effectiveness of prompting techniques within each type since the focus is on understanding the effectiveness of prompting using an LLM for re-ranking. Besides addressing our main question, we also investigate how prompt variations and feature characteristics impact LLM reranking. For prompt variations, we examine the influence of different ranking paradigms, the role of initial document ordering, and the effectiveness of in-context learning. For feature characteristics, we explore the impact of feature groupings (feature source and data type), the number of features, encoding strategies (e.g., float features vs. bucketized features), the importance of feature names, and feature ordering.

Overall, our results paint a differentiated picture. A high-capacity LLM can indeed act as listwise zero-shot reranker for data consisting of hundreds of precomputed ranking features. In terms of its effectiveness though, across both public and private datasets we find that, *on average*, a considerable gap remains, with prompting GPT-4 underperforming compared to the traditional supervised LTR approach LambdaMART by up to 16% wrt. NDCG@10 on the public MSLR-WEB10K dataset. However, when focusing only on the subset of hard samples, i.e. the subset of samples where the LTR approach does rank an *non-relevant* result at the top, we find the LLM-based reranking to improve over the LTR baseline by 22%.

Beyond this main finding, we provide several insights into prompt design choices and their impact on LLM reranking. The results show that using a subset of features often provides better performance than using all features, that LLMs struggle to recover from a poor initial ranking in the listwise case, and that ordering features by their importance is the most reliable strategy compared to reverse or random ordering. However, LLMs remain brittle, with the same strategies sometimes helping or hurting depending on the model size and dataset, highlighting the complexity of this problem. For example, factors such as ranking paradigms, in-context learning, and feature characteristics show mixed results, with their effectiveness varying across different setups.

2 Related Work

The landscape of neural IR research has changed rapidly in the last few years. Initially the focus was on the fine-tuning of the highly effective (but high-latency) cross-encoder models [8, 35, 38], or the less performant but more scalable bi-encoder models [18, 22, 24, 53] for ad-hoc reranking as well as the generation of training data [2, 9, 20] and knowledge distillation [40, 49]. Recently though, attention has shifted towards employing pre-trained LLMs for reranking in a zero-shot manner which does not require any additional fine-tuning or training step: in the *prompt-and-predict* [29] paradigm, all focus is on the prompt and how best to utilize it. A clear advantage of such an approach is little to no need for task-specific training data (i.e. relevance judgments) which can be difficult to obtain in domain-specific scenarios. Existing works have explored various dimensions of this problem which we now discuss in turn. Note that below we only consider approaches that do *not* require additional fine-tuning or training in line with our research focus.

Ranking paradigms. For ranking problems, LLM prompts primarily take one of three forms (*pointwise*, *pairwise* or *listwise*). In the *pointwise* setup the input is one query-document pair and the LLM is tasked with outputting one or more tokens that can be turned into a relevance label; e.g. [14, 46] rank documents by the average log-likelihood of generating the query tokens conditioned on each document with the instruction *Please write a question based on this passage*. A different approach to *query generation* is *relevance generation* where the LLM is tasked with outputting a specific token (e.g. *True* or *False*), and the probability of decoding *True* is used as relevance score [32, 55]. In contrast, *pairwise* rerankers [42] are much more resource-intensive, as the LLM's preference over pairs of documents needs to be repeatedly requested to derive a final ranking. Lastly, in the *listwise* setup, e.g. [32, 39, 40, 49], the LLM prompt contains a query and a list of documents, and the model is tasked with generating an ordered list of document identifiers from most to least relevant wrt. the query (*permutation generation*). As LLMs may have a limited context window, this setup is often combined with a sliding window approach (also called *progressive reranking*): instead of reranking the top-k documents at once, partial lists are extracted, ranked and subsequently combined in a final ranking [32, 49, 55]. In a zero-shot unsupervised setting, for the best-in-class LLMs listwise approaches outperform pointwise approaches but they remain less performant than supervised approaches.

Zero-shot vs. in-context learning. Beyond the query and document information, additional context can be provided in an LLM prompt: demonstrations of (successfully) completed tasks [13] which provide the LLM with additional information about the label space and the distribution of the input tokens as well as examples of the correct input and output formats [33]. In IR, the zero-shot setting has been more popular [40, 46, 54] where beyond the item information and instruction no additional context is provided. One exception is [14] who show that a small number (4) of cleverly chosen demonstrations can be effective in a pointwise ranking setup, outperforming the zero-shot setup by 2-5% depending on the MS MARCO query set and Flan-T5 model employed. At the same time however, the results on several BEIR datasets were less consistent.

Structured data. While the utility of structured data (or more concretely precomputed ranking features) for zero-shot LLM-based rerankers has not been extensively explored in the context of IR,

[1]To our knowledge, the only other LTR dataset with named features is MSLR-WEB30K, of which MSLR-WEB10K is a sample.
[2]We provide empirical evidence for this claim in Table 5.

a few works have considered structured data for other tasks, including text-to-SQL, table question answering, cell lookup, and fact verification [5, 12, 21, 26, 48]. In particular, Sui et al. [48] found that LLMs possess basic structural understanding capabilities but are far from perfect. They demonstrated that carefully chosen input designs, such as the input format and content order, improve LLM performance on various tabular tasks. They also found that adding just one demonstration to the prompt significantly enhances performance across all tested tabular tasks. Furthermore, Jiang et al. [21] proposed an invoking-linearization-generation procedure where the structured data is turned into sentences, and then appended to an LLM's prompt before generating the task output.

Datasets. The widely used MS MARCO passage dataset in combination with the TREC Deep Learning query sets [6, 7] has repeatedly been shown to be amenable to zero-shot LLM-based top-100 reranking. Sun et al. [49] for instance reported a nearly 50% improvement in NDCG@10 over BM25 when employing GPT-4-based listwise prompting. An earlier work [32] using GPT-3 reported a 31% improvement over BM25 on the same dataset. Next to MS MARCO, several other corpora available in the BEIR benchmark are commonly employed [32, 42, 49, 55], but as already pointed out in Section 1, these corpora are as unstructured as MS MARCO.

3 Reranking using LLMs

3.1 Zero-shot reranking

We now describe the different prompt instantiations (cf. Figure 1) we employ in our experiments:

pointwise.yes_no For the pointwise prompt, we follow the approach of [28, 32, 36, 55] to determine the relevance score of each document. We task the LLM with generating whether the document is relevant by outputting either `YES` or `NO`. We then extract the logits of only the last generated token and consider the probability of generating `YES` as the document's relevance score. To score n documents, n prompts are created.

pairwise.sliding-k An exhaustive approach where all pairs of top-n documents for a given query are compared requires $O(n^2)$ prompts. As this is extremely inefficient, we follow the pairwise sliding window approach proposed by Qin et al. [42] as it showed similar effectiveness to the more computationally heavy scoring of all possible document pairs and aggregating the wins to compute an overall ranking. Effectively, we run the first pass of the bubble sort algorithm $k \ll n$ times but always starting at the back of the ranking; this requires $O(k \times n)$ prompts. Given an initial ranking $R = [d_1, d_2, \ldots, d_i, \ldots, d_n]$, we start with the last ordered pair (d_{n-1}, d_n) and prompt the LLM for its preference, randomizing the order of the two documents for each prompt. We order the document pair according to the expressed preference in place and continue with (d_{n-2}, d_{n-1}).

listwise.generation In this setting the entire document list is added to the prompt, and the LLM is instructed to return the best possible reordering, and thus requiring only a single prompt, making it the most efficient of the three approaches. Given the substantial context capacities of our selected LLMs, splitting our rankings into multiple subrankings is unnecessary. At the same time, we note that smaller LLMs (at most a few billion parameters) often struggle with this more complex prompt, outputting incomplete lists or incorrectly formatted lists [42].

Pointwise

I will provide you with a JSON containing a search result item together with a set of features for that item. Based on the features you should decide if the item is relevant. Use the features provided to determine the relevance of the item.
JSON: `{json}`

```
{
    "QualityScore2": 0.75,
    "max of tf*idf - title": 2.1,
    ...
}
```

Is the search item relevant to the query? Answer with YES or NO, nothing else. MAKE SURE the output is either YES or NO.

Pairwise

I will provide you with a JSON containing 2 search result items together with a set of features for each item. Each item has an identifier ('1' or '2') as key. Use the features provided to determine the relevance of the items.
JSON 1: `{json1}`
JSON 2: `{json2}`
Which search item is more relevant to the query? Only respond with the identifier of the more relevant item. MAKE SURE the output is only the numerical identifier.

Listwise

I will provide you with a JSON containing `{num}` search result items together with a set of features for each item. Each item has a numerical identifier as key. Rank the result items based on their relevance. Use the features provided to determine the relevance of each item.
JSON: `{json}`
Rank the `{num}` search result items above based on their relevance. All the result items should be included and listed using identifiers, in descending order of relevance. The output format should be [] > [] > [], e.g., [4] > [0] > [2] > [1]. Only respond with the ranking results, do not add any word or explain. MAKE SURE the output format is [] > [] > [] .. and that all identifiers are present in the list.

Figure 1: Zero-shot prompting: pointwise takes a single document as input, pairwise a pair and listwise a list of documents. Each document is represented by its key-value pairs; keys are the feature names (cf. concrete examples). In the feature space, the query-document similarity is already encoded and thus the prompts contain no textual query.

For both *listwise.generation* and *pairwise.sliding-k* we need to establish a specific initial ranking of the documents we want the LLM to rerank. A typical approach is to consider the ranking order established by a cheaper ranker such as BM25 [45].

Lastly, we need to consider how the precomputed ranking features are represented. Previous research on structured data reasoning with GPT has shown that representing structured data in

markup languages, such as HTML or JSON, outperforms representations in natural language with separators [48]. Based on these findings, we have chosen JSON as the format for our data. Furthermore, we explore different methods to encode numerical features in the prompt, such as using textual buckets (e.g., low, mid, high) or numerical buckets. In addition, we investigate the selection of a subset of features to include and consider how the order of feature presentation can be adjusted to optimize performance.

3.2 In-context learning

Adding demonstrations of successful predictions has been shown to improve pointwise reranking for unstructured data [14]. We extend this to the structured data case by randomly selecting training samples and adding one relevant and one non-relevant item to the *pointwise.yes_no* prompt. These examples provide the model with contextual guidance to distinguish between relevant and non-relevant items, enabling us to evaluate whether in-context learning improves performance in pointwise reranking tasks that utilize precomputed ranking features.

4 Experimental Setup

4.1 Datasets

As discussed in Section 1, to answer our research question we require LTR datasets that contain information about the computed features. However, this information is rarely provided; for this reason we experimented with MSLR-WEB10K [41] and two proprietary LTR datasets derived from the search logs of a large audio streaming service. Incorporating proprietary datasets allows us to evaluate performance on data that we can confidently say the LLMs did not have access to during pretraining.

4.1.1 MSLR-WEB10K. The dataset consists of 10k web search queries and extracted feature vectors from query-url pairs—for each query, on average 120 documents are available—along with manually graded relevance judgments on a 5-point scale. The 136-dimensional feature vector contains a range of standard LTR features (all of them floats) that cover: (i) 13 **document features** such as *SiteRank* and *QualityScore*), (ii) 120 **query-doc features** including *LMIR.DIR* and *BM25* across different parts of the document including the body, anchor and title, and (iii) 3 **interaction features** such as *Query-url click count*. Although some feature names—such as *LM.DIR*—may not appear intuitive, we hypothesize that a large pre-trained LLM has enough exposure to IR documentation in the pre-training dataset to interpret those feature names.

MSLR-WEB10K is preprocessed into five folds, and we used Fold 1 for our experiments. We used the training sets (S1-S3), along with the validation set (S4), to train and validate the supervised LTR baseline model, as described in more detail in Section 4.3. The test set (S5) was used for evaluation and in our LLM experiments. It consists of 2,000 queries. For each query, we limited our experiments to the top 20 ranked documents, as determined by the *BM25 - Title* feature. This feature achieves the highest NDCG@10 among all individual features when evaluated on the validation set (S4). Table 1 presents the distribution of relevance labels for these top-20 ranked documents.

Table 1: Distribution of relevance labels 1 to 4 across the top-20 documents ranked according to the MSLR-WEB10K feature *BM25 - Title* (test set S5 in Fold 1).

Relevance Label	1	2	3	4
Mean # in Top-20	6.92	3.50	0.48	0.21

4.1.2 Audio24 and Audio24-Hard. The MSLR-WEB10K dataset was released more than 10 years ago and may not represent "modern" LTR datasets too well. Therefore, we also consider proprietary datasets (train, validation and two test sets) from a large audio streaming platform, consisting of search logs enriched with relevance labels extracted from user interactions (an item is considered relevant if it was streamed by the user issuing the query). Each item is represented by 300 features: 17 textual (e.g. *entity name* and *entity type*) and 283 numerical features (e.g. *popularity* and *artist follower count*). These features are a subset of the full set employed in the audio streaming platform's search system.

For some analyses, we used the three dedicated feature groupings already listed before: **document features**, **query-doc features**, and **interaction features**. For each grouping, we selected the twenty most important features of that type from the overall feature set (based on feature importance ranking, cf. Section 4.3).[3]

We collected the data in April 2024; it is a random selection of queries issued by US-based users that had a stream success within the top-7[4] items as ranked by the production ranker. The collected data was segmented into training and validation sets, comprising 90k and 22k queries (and their top-7 ranked items), respectively, adhering to an 80/20 split. We used this data to train a supervised LTR model, used as a baseline in our experiments (see Section 4.3).

We compiled two distinct test sets—Audio24 and Audio24-Hard—with 1000 queries each. While Audio24 consists of randomly drawn queries (and their top-7 ranked items and stream successes), Audio24-Hard consists only of queries where the success items were ranked *below* the top position. As the ranker of the production system is already of high quality—due to years of improving and experimenting on the ranker(s)—the Audio24-Hard dataset specifically evaluates the effectiveness of zero-shot LLM rerankers on those challenging queries that the current production ranker does not handle optimally. Another reason for this choice is the potential bias an LLM reranker might exhibit towards reranking: for the Audio24 dataset, the best course of action for a listwise reranker that receives the initial document ordering of the production ranking may be not to rerank the top-ranked documents at all.

We note that we did not compile a separate "hard" test set for MSLR-WEB10K to keep the folds intact and aid reproducibility. However, in Section 5.2 we perform an analysis on the subset of those queries where the LambdaMART baseline did retrieve a non-relevant document at the top rank.

4.2 LLMs

In our experiments, we evaluated zero-shot reranking using two distinct LLMs, both equipped with a context window of 128k tokens:

[3]We resorted to this sampling of features to avoid releasing information on the distribution of feature types in these proprietary datasets.
[4]On almost all mobile devices the top-7 ranked items appear above the fold.

OpenAI's GPT-4 (specifically gpt-4-turbo-2024-04-09[5]) [37], and Meta's open-source Llama-3.1 (specifically Llama-3.1-8B-Instruct) [17], a smaller 8-billion-parameter model that has been pre-trained and instruction-fine-tuned on both natural and synthetic data.

We selected a high-capacity proprietary and an open-source modestly-sized model to represent two ends of the spectrum that practitioners might consider. We do not directly compare GPT-4 and Llama-3.1. Instead, we evaluate prompting using each model separately, comparing the effectiveness against the supervised LambdaMART baseline. By focusing on individual comparisons to LambdaMART, we assess the capabilities of each model in isolation.

The large context window is essential for our experiments, particularly for listwise reranking, which has significant token requirements. For example, on the MSLR-WEB10K dataset, including 20 documents in the prompt results in a prompt size nearing 40k tokens. Similarly, for the Audio24 datasets, adding just 7 documents to the prompt requires approximately 30k tokens.

4.3 Baseline LTR Approach

Today's large-scale search systems typically rely on LTR in the later stages of a multi-stage retrieval pipeline. LambdaMART [3] is a state-of-the-art feature-based LTR approach that is widely deployed. Compared to LLMs, LambdaMART's approach is significantly more efficient in terms of computational resources and training time. LambdaMART is effectively implemented in the LightGBM package [23], as demonstrated by Qin et al. [43] who showed in 2021 that neural LTR lagged behind "traditional" LTR approaches.

We trained LambdaMART models using all available float features (i.e., all available features for MSLR-WEB10K and 283 available features for the Audio24 datasets) with the following hyperparameter settings: num_leaves: 31, learning_rate: 0.05, feature_fraction: 0.9, bagging_fraction: 0.8, bagging_freq: 5, and steps: 100.

For some of our analyses, we also utilized LightGBM's *feature importance ranking*, which is based on the number of times a feature is used to split data during the creation of the decision trees.

4.4 Metrics

For the MSLR-WEB10K dataset we used the commonly employed NDCG@10 [19] metric which is particularly suitable here given the graded relevance judgments. In contrast, Audio24 and Audio24-Hard contain binary labels (stream success or not) and most often (>80%) only a single success item exists in the top-k ranking. Thus, we rely on Mean Reciprocal Rank (MRR). Due to the proprietary nature of the datasets, we do not report absolute metric numbers but instead we present the metrics' *percentage change* from the baseline LTR model LambdaMART as also done in other works involving proprietary industry data, e.g. [34, 47, 51].

5 Results

Recall, that our main question is to what extent the findings of zero-shot LLM rerankers established on plain text corpora hold for datasets consisting of hundreds of precomputed ranking features. We answer this question in Sections 5.1 and 5.2, and present our main results in Table 2. Our further analyses of the problem space are divided into two areas: (i) the impact prompt variations have on an LLM's effectiveness as zero-shot rereanker (Section 5.3) and (ii) the impact feature characteristics have (Section 5.4).

5.1 Can zero-shot LLM rerankers deal with precomputed ranking features?

The high-capacity model GPT-4 can indeed rerank documents represented by hundreds of precomputed ranking features in a zero-shot manner, but the effectiveness gap to the supervised baseline remains high (Table 2). For the MSLR-WEB10K dataset, the GPT-4-based reranking performs 4.0% better than a pure lexical reranker (cf. rows 2a/G3c) and 11.4% better than a random reordering of the top-20 documents (cf. rows 2b/G3c). At the same time, GPT-4-based reranking performs 15.6% worse than our supervised LTR baseline (cf. rows 1a/G3a). For the Audio24 dataset, we observe similar trends with GPT-4-based reranking performing 8.6% worse than LambdaMART (cf. rows 1a/G3a).

Turning to zero-shot reranking with the open-source model Llama-3.1, it demonstrates the ability to extract useful information from the features, achieving a 5.3% improvement over a randomly ordered top-20 baseline for MSLR-WEB10K (cf. rows 2b/L3c). On the Audio24 dataset, Llama-3.1-based reranking performs significantly worse than the supervised LambdaMART baseline (-26.5%, cf. row L3c).

5.2 Do LLMs improve rankings on hard queries?

GPT-4- and Llama-3.1-based reranking outperform LambdaMART in challenging scenarios like Audio24-Hard, where traditional supervised rankers struggle. The Audio24 dataset represents a general sampling of queries, while Audio24-Hard is specifically constructed from cases where the production ranker failed to place the most relevant item at the top. This makes Audio24-Hard valuable for evaluating the potential of LLM-based rerankers in scenarios with room for improvement. Our experiments show that LLMs can capitalize on this opportunity, with GPT-4-based reranking outperforming the LambdaMART baseline by 4.5% on Audio24-Hard (cf. rows 1a/G5b). This demonstrates the ability of LLMs to provide meaningful reranking in more challenging cases.

We conducted a similar evaluation with the MSLR-WEB10K dataset. By filtering out cases where LambdaMART placed a non-relevant item at the top, we created a subset of the test set (S5, cf. Section 4.1.1) that mimics the challenges of Audio24-Hard ($n = 566$). On this filtered set, GPT-4-based reranking achieved an NDCG@10 score of 0.431, compared to 0.354 for LambdaMART. This again confirms that LLMs excel when traditional ML-based rerankers fail.

In addition, we conducted a human evaluation on Audio24-Hard, comparing the rankings from GPT-4 (cf. row G5b) with those of LambdaMART (cf. row 1a). Two annotators independently assessed 200 rankings, choosing between "Ranking 1," "Ranking 2," or "Don't Know." To ensure the evaluation reflected a realistic user-facing scenario, the rankings were presented using a subset of string features visible to users in the mobile app of the audio streaming service.

The results show a consistent preference for GPT-4-based rerankings, which are favored in ~30% of cases, compared to less than 20% for LambdaMART. A notable proportion of cases (~50%) fall into the "Don't Know" category, highlighting the complexity of the

[5]Initial experiments were conducted in May 2024, and for consistency and comparability across all experiments, we used the same GPT version throughout the study.

Table 2: Results shown for the high-capacity GPT-4 and the modestly-sized Llama-3.1 8B model. The results should be interpreted only within each model's section. *Prev* refers to the first-stage ranking: (i) BM25-title score for MSLR-WEB10K, (ii) production ranking for Audio24 datasets, and (iii) a random ordering within the top-k. For MSLR-WEB10K and Audio24, results are stat. significant compared to the baseline (row 1a) using a pairwise t-test. For Audio24-Hard, significance is denoted by * for p<0.05 and ** for p<0.001. Baseline values for Audio24 and Audio24-Hard are proprietary, so results are shown as percentage changes.

			Method	Prev	Data Type	Feature Source	MSLR10K (k: 20) NDCG@10	Audio24 (k: 7) Rel MRR@7	Audio24-Hard (k: 7) Rel MRR@7
			Supervised Methods						
		(1a)	LambdaMART	–	Float	All	**0.648**	***private***	*private*
		(1b)	LambdaMART	–	Float	Document	0.587	-13.2 %	**+0.02 %**
		(1c)	LambdaMART	–	Float	Query-Doc	0.587	-17.9 %	-0.02 %
		(1d)	LambdaMART	–	Float	Interaction	0.554	-3.49 %	-0.03 %
			Zero-shot Methods						
		(2a)	BM25	–	–	–	0.526	–	–
		(2b)	Random	–	–	–	0.491	–	–
GPT-4	**Data Type**	(G3a)	*listwise.generation*	Random	Float	–	0.544	**-8.62 %**	+0.50 %
		(G3b)	*listwise.generation*	BM25/Prod	Float	–	0.532	-28.9 %	+4.47 %*
		(G3c)	*pairwise.sliding-4*	BM25/Prod	Float	–	**0.547**	-21.5 %	-2.20 %
		(G3d)	*pointwise.yes_no*	–	Float	–	0.540	-14.9 %	+0.54 %
		(G4a)	*listwise.generation*	Random	String	–	–	-17.9 %	+1.34 %
		(G4b)	*listwise.generation*	Prod	String	–	–	-9.28 %	+2.93 %
		(G4c)	*pairwise.sliding-4*	Prod	String	–	–	-21.1 %	+1.04 %
		(G4d)	*pointwise.yes_no*	–	String	–	–	-22.2 %	+1.08 %
		(G5a)	*listwise.generation*	Random	All	–	–	-9.21 %	+2.71 %
		(G5b)	*listwise.generation*	Prod	All	–	–	-11.1 %	**+4.51 %***
		(G5c)	*pairwise.sliding-4*	Prod	All	–	–	-22.8 %	+1.68 %
		(G5d)	*pointwise.yes_no*	–	All	–	–	-20.8 %	+1.49 %
	Feature Source	(G6a)	*listwise.generation*	Random	Float	Document	0.515	-23.1 %	**+4.43 %***
		(G6b)	*listwise.generation*	BM25/Prod	Float	Document	0.518	-25.3 %	+4.16 %
		(G6c)	*pairwise.sliding-4*	BM25/Prod	Float	Document	0.513	-18.8 %	+3.81 %
		(G6d)	*pointwise.yes_no*	–	Float	Document	0.511	-21.8 %	+0.97 %
		(G7a)	*listwise.generation*	Random	Float	Query-Doc	0.532	-20.4 %	-0.70 %
		(G7b)	*listwise.generation*	BM25/Prod	Float	Query-Doc	0.530	-19.7 %	-0.67 %
		(G7c)	*pairwise.sliding-4*	BM25/Prod	Float	Query-Doc	0.539	-23.4 %	-0.71 %
		(G7d)	*pointwise.yes_no*	–	Float	Query-Doc	0.531	-23.7 %	-0.08 %
		(G8a)	*listwise.generation*	Random	Float	Interaction	0.532	-6.47 %	-0.75 %
		(G8b)	*listwise.generation*	BM25/Prod	Float	Interaction	0.544	-6.05 %	+0.51 %
		(G8c)	*pairwise.sliding-4*	BM25/Prod	Float	Interaction	**0.545**	**-3.03 %**	-1.42 %
		(G8d)	*pointwise.yes_no*	–	Float	Interaction	0.532	-11.3 %	+3.33 %
Llama-3.1	**Data Type**	(L3a)	*listwise.generation*	Random	Float	–	0.505‡	-27.7 %	-3.97 %
		(L3b)	*listwise.generation*	BM25/Prod	Float	–	0.507‡	-57.1 %	-23.8 %**
		(L3c)	*pairwise.sliding-4*	BM25/Prod	Float	–	**0.517**	-27.6 %	**+1.97 %**
		(L3d)	*pointwise.yes_no*	–	Float	–	0.496	-46.0 %	-9.45 %*
		(L4a)	*listwise.generation*	Random	String	–	–	-37.3 %	-4.99 %*
		(L4b)	*listwise.generation*	Prod	String	–	–	-51.9 %	-18.1 %**
		(L4c)	*pairwise.sliding-4*	Prod	String	–	–	-32.8 %	+0.08 %
		(L4d)	*pointwise.yes_no*	–	String	–	–	-44.1 %	-7.41 %*
		(L5a)	*listwise.generation*	Random	All	–	–	**-26.5 %**	-2.34 %
		(L5b)	*listwise.generation*	Prod	All	–	–	-52.0 %	-19.6 %**
		(L5c)	*pairwise.sliding-4*	Prod	All	–	–	-29.8 %	-3.81 %
		(L5d)	*pointwise.yes_no*	–	All	–	–	-44.7 %	-5.90 %*
	Feature Source	(L6a)	*listwise.generation*	Random	Float	Document	0.492†	-34.0 %	-3.69 %
		(L6b)	*listwise.generation*	BM25/Prod	Float	Document	0.506†	-54.4 %	-19.9 %**
		(L6c)	*pairwise.sliding-4*	BM25/Prod	Float	Document	0.510	-31.7 %	-0.87 %
		(L6d)	*pointwise.yes_no*	–	Float	Document	0.492	-45.4 %	-2.94 %
		(L7a)	*listwise.generation*	Random	Float	Query-Doc	0.492‡	-37.9 %	-1.82 %
		(L7b)	*listwise.generation*	BM25/Prod	Float	Query-Doc	0.511‡	-55.8 %	-20.6 %**
		(L7c)	*pairwise.sliding-4*	BM25/Prod	Float	Query-Doc	0.522	-32.1 %	-5.10 %*
		(L7d)	*pointwise.yes_no*	–	Float	Query-Doc	0.486	-44.6%	-9.09 %**
		(L8a)	*listwise.generation*	Random	Float	Interaction	0.531†	**-26.2 %**	-0.85 %
		(L8b)	*listwise.generation*	BM25/Prod	Float	Interaction	0.523†	-57.1 %	-22.1 %**
		(L8c)	*pairwise.sliding-4*	BM25/Prod	Float	Interaction	**0.529**	-30.6 %	**-0.83 %**
		(L8d)	*pointwise.yes_no*	–	Float	Interaction	0.511	-44.3 %	-6.52 %*

†: *Runs with less than 80 % successful rankings.* ‡: *Runs with less than 30 % successful rankings.*

task.[6] Annotator agreement stands at 70 %, indicating reasonable consistency. These findings suggest that zero-shot reranking using GPT-4 performs particularly well in scenarios such as Audio24-Hard, where traditional rankers face significant challenges.

5.3 Prompt Variations

5.3.1 What ranking paradigm is effective? When comparing the different ranking paradigms we find GPT-4 to perform best in the *pairwise.sliding-4* setting for MSLR-WEB10K and the *listwise.generation* setting for the Audio24 datasets (cf. rows G3a-c). For Llama-3.1, the *pairwise.sliding-4* prompt is most effective across all datasets (cf. row L3c). **Pointwise reranking generally performs worst. Overall, we cannot point to a clear winning ranking paradigm.**

In line with [42], we find that the modestly-sized Llama-3.1 model struggles to correctly generate permutations of document identifiers for the majority of *listwise.generation* prompts on the MSLR-WEB10K dataset (marked with † and ‡ in Table 2), where we rerank the top-20 documents. However, smaller lists are less problematic as for the Audio24 datasets (top-7 documents), Llama-3.1's success rate for *listwise.generation* prompts exceeds 95%.

5.3.2 Does the initial document ordering matter in the listwise prompt? Previous research on plain text corpora has demonstrated that the initial ranking significantly influences the outcomes in the listwise paradigm, particularly when LLMs are employed [42, 49, 55]. To investigate if this also holds for documents represented by precomputed ranking features we explored two initial rankings: a random reordering of the top-k documents and a BM25-based ordering (MSLR-WEB10K) and a production ranker ordering (Audio24 and Audio24-Hard) respectively. **We find that document ordering indeed impacts the reranking effectiveness but inconsistently:** for MSLR-WEB10K, GPT-4 performs better with a random top-k ordering (row G3a) when prompting it with all available features, but the BM25 ordering leads to the best results when prompting only with the interaction features (row G8b). We observe similar inconsistencies across the two Audio24 datasets. What is consistent is the fact that if we provide an intentionally very poor initial ranking, e.g. the reverse of the production ranking as shown in Figure 2 for GPT-4 (specifically *Listwise-Pre-Rev*), the LLM is not able to recover a good reranking and the effectiveness drops significantly (e.g., for MSLR-WEB10K, from an NDCG@10 of 0.53 to 0.49) compared to an initial high quality ranking (i.e. *Listwise-Pre*).

5.3.3 Does in-context learning help? Next, we investigated the effect of in-context learning, which has demonstrated improved effectiveness over zero-shot methods in pointwise unstructured data reranking [14] and structured data reasoning [48]. Specifically, we tested a two-shot setup: as example we included two query-document pairs with one document being relevant and one non-relevant to the selected example query (randomly selected from the training folds). For each list of documents to score, we used the same two examples. For each new list of documents to score we randomly selected two new examples from the training folds. For MSLR-WEB10K (which has multiple levels of relevance), we considered documents with a relevance label of 1 to 4 as relevant. On the Audio24 dataset, adding in-context examples significantly improved GPT-4-based reranking, increasing the relative MRR@7 score from -14.9% to -5.6% (though still below the baseline's effectiveness). However, no improvements were observed for GPT-4-based reranking on the MSLR-WEB10K and Audio24-Hard datasets. Similarly, Llama-3.1-based reranking did not exhibit any significant improvements across the evaluated datasets, further highlighting the variability of in-context learning's effectiveness with different LLMs. **Overall, our results show that in-context learning can enhance reranking performance, but its effectiveness varies significantly across datasets and LLM models.**

[6]Annotator 1 preferences: GPT-4: 34.5%, LambdaMART: 12%, Don't know: 53%; Annotator 2 preferences: GPT-4: 25.5%, LambdaMART: 18%, Don't know: 56.5%

5.4 Feature Characteristics

So far we focused on the prompt setup, now we explore the impact different feature setups have on reranking effectiveness.

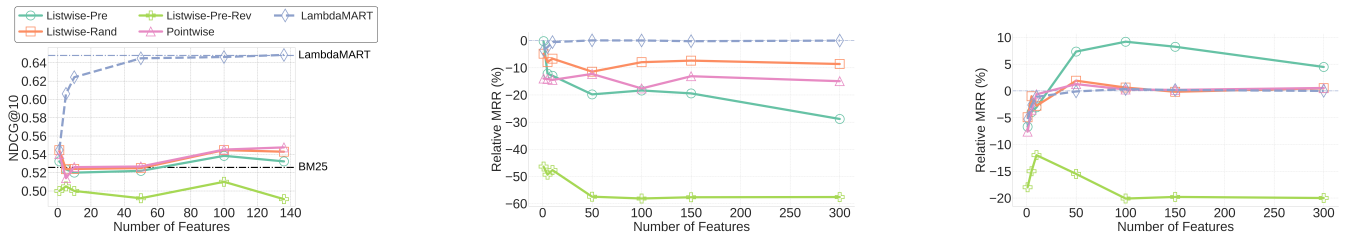
5.4.1 What feature groupings are effective? We here discuss feature groups according to the source they are derived from: either document only, query-document or user interaction features [31]. For the Audio24 datasets we also have features of different data types available (string vs. float) and thus compare those as well.

First, we consider feature sources for LambdaMART. For MSLR-WEB10K, using all available features performs best by a large margin: NDCG@10 of 0.648 when training on all features vs. 0.587 when only training on the document or the query-doc feature groups (cf. rows 1a/b/c). A similar trend holds for the Audio24 dataset, though here training only on interaction features delivers the second best effectiveness (cf. row 1d). The results are more mixed for Audio24-Hard: here using fewer features (document-only, cf. row 1b) actually slightly improves over the model trained on all features. This can be explained by the fact that we trained a single LambdaMART model based on data that has the same distribution as the Audio24 dataset as described in Section 4.3.

When we consider our zero-shot LLM rerankers, the interaction feature group, though only consisting of three features for MSLR-WEB10K (query-url click count, url click count and url dwell time), delivers strong results across the datasets. For GPT-4-based rerankings, on the MSLR-WEB10K dataset, the highest performance is achieved using all available features (NDCG@10 0.547, cf. row G3c), but it is a comparable result to using only interaction features (NDCG@10 0.545, cf row G8c). On the Audio24 dataset, ranking based solely on interaction features leads to better reranking effectiveness compared to using all available features (cf. rows G5a/G8c). For Audio24-Hard, prompting GPT-4 using all features (cf. row G5b) achieves the best performance, followed by using document features alone (cf. row G6a).

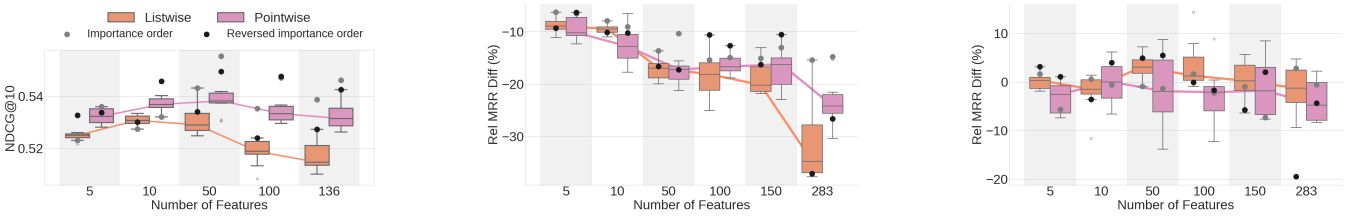
For Llama-3.1-based rerankings, on the MSLR-WEB10K and Audio24 datasets, the interaction feature group delivers the best performance (cf. row L8c). For Audio24-Hard, Llama-3.1 achieves its best performance when prompted with all float features, but with interaction features being the best feature source group.

When investigating the split of features by data types for the Audio24 datasets, we expected string features (like *entity type*, *track name*, *artist name* or *album name*) to be more effective than float features (cf. rows G3*/G4*) given GPT-4's world knowledge and the



(a) MSLR-WEB10K (b) Audio24 (c) Audio24-Hard

Figure 2: Overview of the impact of differently sized feature subsets, with features ordered by their importance (based on a LambdaMART ranking) and the most important features added first. Listwise-Pre refers to GPT-4's *listwise.generation* with BM25 (MSLR-WEB10K) or the production ranker (Audio24, Audio24-Hard) as initial ranking. Listwise-Pre-Rev is the *reversed* ranking of Listwise-Pre.



(a) MSLR-WEB10K (b) Audio24 (c) Audio24-Hard

Figure 3: Impact of feature ordering on ranking performance for three datasets: MSLR-WEB10K (a, NDCG@10), Audio24 (b, Relative MRR@7) and Audio24-Hard (c, Relative MRR@7), using the Listwise-Rand and Pointwise approaches. Each boxplot shows performance across 12 feature orderings, with gray and black dots indicating runs where features are ordered by LambdaMART's importance ranking and its reverse, respectively.

type of queries in the Audio24 and Audio24-Hard datasets (for example: *frank ocean*, *the weeknd*, and *relaxing reggae*). This, however, is not the case, indicating the complexity of the ranking problem.

Table 3: Feature importance ranking for MSLR-WEB10K: list of the five most and least important features.

Most Important	Least Important
Outlink number	Boolean model - body
PageRank	Boolean model - anchor
BM25 - title	Boolean model - title
QualityScore2	Boolean model - url
URL click count	Boolean model - whole document

Table 4: Performance comparison of different feature bucket configurations on MSLR-WEB10K. NDCG@10 scores are reported for float features (cf. rows G3a/b/d in Table 2), textual buckets (LMH: Low, Mid, High), and numerical buckets: (0-2), (0-4), and (0-9). The best-performing bucket configuration for each approach is highlighted in bold.

	LLM	Prev	Float	LMH	(0-2)	(0-4)	(0-9)
listwise.generation	GPT-4	Rand.	**0.544**	0.519	0.513	0.521	0.526
listwise.generation	GPT-4	BM25	0.532	0.526	0.531	0.528	**0.533**
pointwise.yes_no	GPT-4	—	**0.540**	0.521	0.506	0.514	0.528

Table 5: Impact of informative feature names on the MSLR-WEB10K dataset. We report NDCG@10 for the original feature names (*Named*, corresponding to rows G3a/b/d in Table 2) and runs where we replaced the names by numerical IDs (*Integer*). All differences between *Named* and *Integer* are statistically significant based on the paired t-test ($p < 0.001$).

	LLM	Prev	Type	Named	Integer
listwise.generation	GPT-4	Rand.	Float	**0.544**	0.511
listwise.generation	GPT-4	BM25	Float	**0.532**	0.512
pointwise.yes_no	GPT-4	—	Float	**0.540**	0.503

5.4.2 Does the number of features matter? To answer this question we ordered features by their importance wrt. the trained LambdaMART model (as discussed in Section 4.3) and explore their added utility for the rerankers. For MSLR-WEB10K the five most important and least important features can be found in Table 3.

Based on this feature ranking, we retrained LambdaMART with different subsets of features (1, 5, 10, 50, 100, 136 for MSLR-WEB10K and 1, 5, 10, 50, 100, 150, 283 for Audio24 and Audio24-Hard) and prompted GPT-4 with the same feature subsets. The results can be found in Figure 2, where we compare LambdaMART and different GPT-4-based ranking variants: the *pointwise.yes_no* variant (equivalent to Pointwise) and three *listwise.generation* variants: *Listwise-Rand*, *Listwise-Pre*, and *Listwise-Pre-Rev* (initial random ordering of

the top-k documents, ordered by BM25 or the production ranker, and reverse-ordered by BM25 or the production ranker, respectively)[7].

As expected, based on how we derived the feature importance, LambdaMART consistently benefits from the addition of features, reaching 95% of its top effectiveness with just five features. Our unsupervised zero-shot LLM rerankers, however, do not show consistent improvements, even when provided with only the most important features. For MSLR-WEB10K, approximately 50 features are needed to consistently surpass the BM25 baseline (for both *Listwise-Rand* and *Pointwise*). On the Audio24 dataset, depending on the initial ranking, between 100 and 150 features yield the highest effectiveness for *Listwise-Pre* and *Listwise-Rand*, with similar trends observed for Audio24-Hard.

For almost all settings (except for *Pointwise* in Figure 2a), including a subset of the features yields better performance compared to including all features. This indicates the importance of feature selection for LLM reranking. This finding aligns with earlier studies on question-answering tasks using structured data, which also highlight the significance and complexity of feature selection [21, 26]. **Our results suggest that while the number of features is crucial, traditional feature importance approaches may not be particularly suitable for zero-shot LLM rerankers.**

5.4.3 Does bucketizing the features help? One may argue that a feature value like '0.5354' by itself is not informative enough for a zero-shot LLM-based reranker and additional *context* of how to interpret the value (e.g. is it particularly low or high; is it bounded) is likely to lead to a more effective reranker. We thus examined the impact of encoding features into different bucket configurations, as shown in Table 4: we compare the direct use of float features with the use of textual buckets where each float feature is converted into a text label *low*/*medium*/*high*, depending on the feature value and the use of numerical buckets where each feature value is converted into an integer either 0-2, 0-4 or 0-9 respectively. To ensure the LLM could interpret the bucketed values, the prompt was also updated to include a description of the bucketing strategy, explaining what each bucket represents. For MSLR-WEB10K, the results indicate that GPT-4 generally performs better when prompted with float features directly, achieving the highest NDCG@10 scores for *listwise.generation*, with random ordered input (0.544, cf. row G3a) and *pointwise.yes_no* (0.540, cf. row G3d). For *listwise.generation* with BM25 ranked input, however, bucketization with 10 numerical buckets slightly outperforms float features, achieving an NDCG@10 of 0.533 compared to 0.532 for float features (cf. row G3b). We observe no improvements on either of the Audio24 datasets. Overall, **bucketization—whether textual or numerical—typically leads to lower performance for most configurations compared to directly using float features.**

5.4.4 Are the feature names important? To evaluate the importance of feature names, we replaced feature names by numerical identifiers. Table 5 demonstrates that GPT-4 performs worse when using numerical identifiers compared to named features (i.e., NDCG@10 0.544 with named vs. 0.511 with identifiers for *listwise.generation*) but still better than random (0.491, cf. row 2b) on the MSLR-WEB10K dataset. **Thus, our results show that feature names are crucial when GPT-4 is prompted in a zero-shot manner using precomputed ranking features.**

5.4.5 Does the feature ordering matter? So far, we have varied document ordering and the number of features. Next, we explore how ordering input features *within* a document affects reranking performance.

We focused GPT-4-based reranking and used as initial documents ordering a random, but fixed, order of the top-k documents (i.e. each query has a fixed random order of the documents). The same feature subsets as in Section 5.4.1 were used. Once a feature subset was fixed, each prompt was executed twelve times with a different ordering of the features within a document: ten random feature orders (kept consistent across documents), one ordering based on feature importance, and one based on the *reverse* feature importance. Due to the large number of prompts, this analysis was conducted on a sample of 500 queries for MSLR-WEB10K and 100 queries each for Audio24 and Audio24-Hard.

The results, summarized in Figure 3, reveal one clear signal: **overall, it appears best to rank the features in the order of their importance**, with the most important features appearing first within a document. In 22 out of 34 cases—aggregated across all datasets and both pointwise and listwise rankings—the effectiveness of the importance-based feature ordering exceeded 75% of the random reorderings, as illustrated by the position of the gray *importance order* marker above the upper quartile lines.

6 Conclusions

We set out to explore to what extent the findings of zero-shot LLM rerankers established on plain text corpora hold for datasets containing predominantly or exclusively precomputed ranking features. We conducted experiments on a public LTR dataset and two proprietary datasets collected from an audio streaming platform's search logs. On average across both public and private datasets we find that zero-shot LLM-based rerankers do not outperform a high-quality learning-to-rank baseline such as LambdaMART (and in fact significantly underperforms it). However, for the subset of hard queries (i.e. queries where the LTR approach fails to place a relevant document at rank 1), we do find zero-shot ranking using a high-capacity LLM such as GPT-4 to be outperforming LambdaMART by up to 4.5%. We also explored what impact different prompt and feature choices have on the LLM-based reranker's effectiveness. We found that different design choices often lead to inconsistent results and that it takes considerable effort to arrive at settings that consistently yield improvements. Overall, we have to conclude that the vision of simply prompting an LLM with all available precomputed features is premature.

Our findings guide the way to future work: as zero-shot LLM rerankers outperform LTR on retrospectively found to be hard queries, query performance prediction may be a way towards combining an LTR and LLM-based ranking setup. Given the impact different feature choices have, automatic prompt optimization [11] is another avenue to explore. And lastly, we need to address the shortage of LTR datasets that can act as benchmarks to further investigate LLM-based rerankers.

[7] *pairwise.sliding-4* was excluded from the deep-dive experiments due to its significantly higher computational cost.

References

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

[2] Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Unsupervised dataset generation for information retrieval. In *SIGIR 2022*. 2387–2392.

[3] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[4] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*. PMLR, 1–24.

[5] Wenhu Chen. 2023. Large Language Models are few(1)-shot Table Reasoners. In *Findings of EACL 2023*. 1120–1130.

[6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 deep learning track. (2020).

[7] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2019. Overview of the TREC 2019 deep learning track. *arXiv preprint arXiv:2003.07820* (2019).

[8] Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for IR with contextual neural language modeling. In *SIGIR 2019*. 985–988.

[9] Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith Hall, and Ming-Wei Chang. 2023. Promptagator: Few-shot Dense Retrieval From 8 Examples. In *ICLR 2023*.

[10] Domenico Dato, Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, and Nicola Tonellotto. 2022. The Istella22 dataset: Bridging traditional and neural learning to rank evaluation. In *SIGIR 2022*. 3099–3107.

[11] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning. In *EMNLP 2022*. 3369–3391.

[12] Haoyu Dong and Zhiruo Wang. 2024. Large Language Models for Tabular Data: Progresses and Future Directions. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. 2997–3000.

[13] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[14] Andrew Drozdov, Honglei Zhuang, Zhuyun Dai, Zhen Qin, Razieh Negin Rahimi, Xuanhui Wang, Dana Alon, Andrew McCallum, Mohit Iyyer, Don Metzler, and Kai Hui. 2023. PaRaDe: Passage Ranking using Demonstrations with Large Language Models. In *Findings of EMNLP 2023*.

[15] Guglielmo Faggioli, Laura Dietz, Charles LA Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, et al. 2023. Perspectives on large language models for relevance judgment. In *ICTIR 2023*. 39–50.

[16] Naghmeh Farzi and Laura Dietz. 2024. Pencils down! automatic rubric-based evaluation of retrieve/generate systems. In *Proceedings of the 2024 ACM SIGIR International Conference on Theory of Information Retrieval*. 175–184.

[17] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 https://arxiv.org/abs/2407.21783

[18] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022).

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

[20] Vitor Jeronymo, Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, Roberto Lotufo, Jakub Zavrel, and Rodrigo Nogueira. 2023. Inpars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820* (2023).

[21] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *EMNLP 2023*. 9237–9251.

[22] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP 2020*. 6769–6781.

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[24] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *SIGIR 2020*. 39–48.

[25] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466.

[26] Younghun Lee, Sungchul Kim, Tong Yu, Ryan A Rossi, and Xiang Chen. 2024. Learning to Reduce: Optimal Representations of Structured Data in Prompting Large Language Models. (2024). arXiv:2402.14195

[27] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-based product retrieval in taobao search. In *KDD 2021*. 3181–3189.

[28] Percy Liang, Rishi Bommasani, Tony Lee, et al. 2023. Holistic Evaluation of Language Models. *Transactions on Machine Learning Research* (2023).

[29] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[30] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade ranking for operational e-commerce search. In *KDD 2017*. 1557–1565.

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

[32] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-Shot Listwise Document Reranking with a Large Language Model. arXiv:2305.02156

[33] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?. In *EMNLP 2022*. 11048–11064.

[34] Bhaskar Mitra and Nick Craswell. 2015. Query auto-completion for rare prefixes. In *CIKM 2015*. 1755–1758.

[35] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[36] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Findings of EMNLP 2020*. 708–718.

[37] OpenAI, Josh Achiam, Steven Adler, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[38] Aleksandr V. Petrov, Sean MacAvaney, and Craig Macdonald. 2024. Shallow Cross-Encoders for Low-Latency Retrieval. In *ECIR 2024*. Springer, 151–166.

[39] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. arXiv:2309.15088

[40] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! arXiv:2312.02724

[41] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[42] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2024. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. arXiv:2306.17563

[43] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2021. Are Neural Rankers still Outperformed by Gradient Boosted Decision Trees?. In *ICRL 2021*.

[44] Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R Hersh. 2021. Searching for scientific evidence in a pandemic: An overview of TREC-COVID. *Journal of Biomedical Informatics* 121 (2021), 103865.

[45] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[46] Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving Passage Retrieval with Zero-Shot Question Generation. In *EMNLP 2022*. 3781–3797.

[47] Yi Su, Xiangyu Wang, Elaine Ya Le, Liang Liu, Yuening Li, Haokai Lu, Benjamin Lipshitz, Sriraj Badam, Lukasz Heldt, Shuchao Bi, Ed H. Chi, Cristos Goodrow, Su-Lin Wu, Lexi Baugher, and Minmin Chen. 2024. Long-Term Value of Exploration: Measurements, Findings and Algorithms. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM '24)*. 636–644.

[48] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *The 17th ACM International Conference on Web Search and Data Mining (WSDM '24)*.

[49] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In *EMNLP 2023*. 14918–14937.

[50] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *NeurIPS 2021*.

[51] Paul Thomas, Seth Spielman, Nick Craswell, and Bhaskar Mitra. 2024. Large Language Models can Accurately Predict Searcher Preferences. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 1930–1940.

[52] Yuan Wang, Peifeng Yin, Zhiqiang Tao, Hari Venkatesan, Jin Lai, Yi Fang, and PJ Xiao. 2023. An empirical study of selection bias in pinterest ads retrieval. In *KDD 2023*. 5174–5183.
[53] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *ICLR 2021*.
[54] Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Bendersky. 2024. Beyond Yes and No: Improving Zero-Shot LLM Rankers via Scoring Fine-Grained Relevance Labels. arXiv:2310.14122
[55] Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2024. A Setwise Approach for Effective and Highly Efficient Zero-shot Ranking with Large Language Models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 38–47.