Set-Encoder: Permutation-Invariant Inter-Passage Attention for Listwise Passage Re-Ranking with Cross-Encoders

Ferdinand Schlatt
Friedrich-Schiller-Universität Jena

Maik Fröbe
Friedrich-Schiller-Universität Jena

Harrisen Scells
Leipzig University

Shengyao Zhuang
CSIRO

Bevan Koopman
CSIRO

Guido Zuccon
University of Queensland

Benno Stein
Bauhaus-Universität Weimar

Martin Potthast
University of Kassel, hessian.AI, and ScadDS.AI

Matthias Hagen
Friedrich-Schiller-Universität Jena

ABSTRACT

Cross-encoders are effective passage re-rankers. But when re-ranking multiple passages at once, existing cross-encoders inefficiently optimize the output ranking over several input permutations, as their passage interactions are not permutation-invariant. Moreover, their high memory footprint constrains the number of passages during listwise training. To tackle these issues, we propose the Set-Encoder, a new cross-encoder architecture that (1) introduces inter-passage attention with parallel passage processing to ensure permutation invariance between input passages, and that (2) uses fused-attention kernels to enable training with more passages at a time. In experiments on TREC Deep Learning and TIREx, the Set-Encoder is more effective than previous cross-encoders with a similar number of parameters. Compared to larger models, the Set-Encoder is more efficient and either on par or even more effective.

CCS CONCEPTS

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

KEYWORDS

Cross-encoder; Re-ranking; Listwise learning to rank; Permutation invariance; Inter-passage attention; Fused-attention kernel

1 INTRODUCTION

Cross-encoders [1, 49, 81] using pre-trained transformer-based language models are among the most effective re-rankers [36, 65]. By building a semantic representation of a query and a passage, they circumvent the vocabulary-mismatch problem during relevance scoring [40]. However, when re-ranking multiple passages at once, the existing cross-encoders lack a central desirable property of learning-to-rank models [52]: permutation-invariant interactions between the input passages. Interactions between passages can improve the semantic representations through information exchange, and permutation invariance ensures the same ranking is output independent of the ordering of the input passages.

Several cross-encoders, such as 'duo' models [51, 57] for two-passage inputs or decoder-only LLMs for up to 20 passages [58, 59, 69], model passage interactions when re-ranking many of them at once. However, these models all just concatenate the input passages

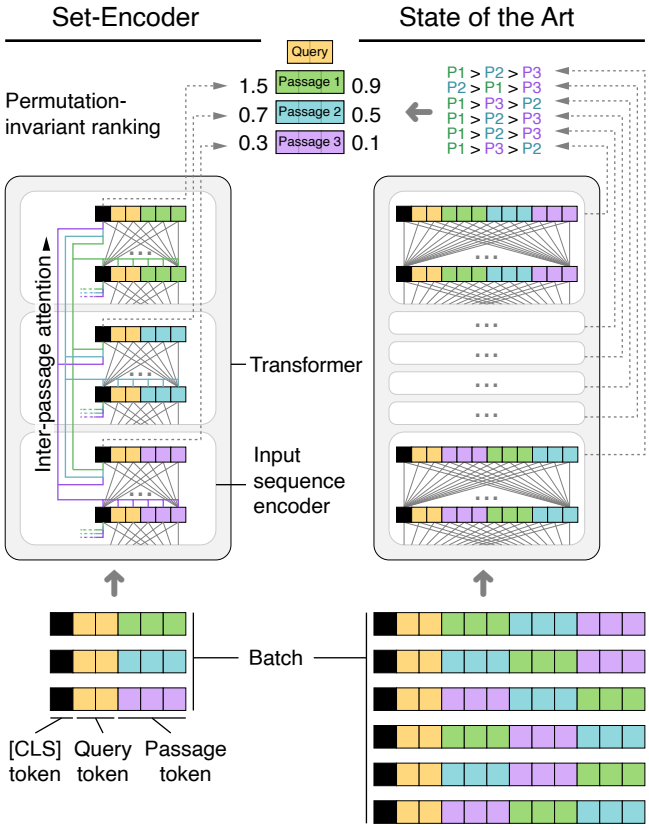

Figure 1: Architecture comparison of our new Set-Encoder with previous state-of-the-art re-rankers that model passage interactions, exemplified by three passages. Previous re-rankers concatenate the input passages, leading to potentially inconsistent rankings between different passage permutations so that many (or all) permutations are re-ranked for optimization. The Set-Encoder avoids input concatenation and instead uses a novel inter-passage attention pattern that makes the re-ranking process permutation-invariant.

into a single sequence, so that the derived re-rankings frequently depend on the order of the input passages. Figure 1 (right) illustrates that a solution then usually is to re-rank multiple input permutations to optimize the final re-ranking's effectiveness. There are heuristics to reduce the number of permutations [32, 69, 72], but re-rankers that directly output the same ranking, irrespective of the order of the input passages, would be more efficient.

In this paper, we propose the *Set-Encoder*,[1] a new cross-encoder architecture that models passage interactions in a permutation-invariant way (see Figure 1, left). Instead of concatenating the input passages, the Set-Encoder processes them in parallel as different sequences in a batch. To enable passage interactions, the Set-Encoder lets each sequence aggregate information in a single special [CLS] embedding token that all other sequences can attend to. The model is permutation-invariant because all [CLS] tokens share the same positional encoding. We call our new attention pattern *inter-passage attention* (Section 3).

Besides their input permutation dependence, another issue of current transformer-based cross-encoders is that their memory inefficiency limits the number of passages per query during training or fine-tuning—to our knowledge, 36 is the highest number reported [84]. For the Set-Encoder, we alleviate the memory issue by using fused-attention kernels [39] that enable fine-tuning on and re-ranking of 100 passages per query. For the actual fine-tuning, we further revisit the observation that cross-encoders, which are fine-tuned on the "standard" MS MARCO with heuristically sampled (i.e., noisy) hard negatives, are less effective than distilled cross-encoders, which are fine-tuned on LLM-based rankings [69, 71]. But as the available distillation datasets have shortcomings (e.g., only 20 passages per query [69]), we create an improved distillation dataset with 100 passages per query using more recent LLMs, and fine-tune our Set-Encoder in a two-stage process. First, we fine-tune on the large but noisy MS MARCO data, and then continue on our smaller but high-quality distillation data (Section 4).

Our experiments on the TREC Deep Learning tracks [23, 24] and the TIREx platform [30] demonstrate that the Set-Encoder improves effectiveness over a cross-encoder without inter-passage attention, especially in out-of-domain scenarios. Compared to existing cross-encoders, the Set-Encoder is similarly effective but with substantially fewer parameters. Furthermore, due to its permutation invariance, the Set-Encoder does not need to re-rank multiple permutations of a set of passages and thus is more efficient at inference time than existing listwise cross-encoders (Section 5).

2 RELATED WORK

In this section, we review the benefits of passage interactions and permutation invariance in cross-encoders, the techniques for fine-tuning them, the limitations of these techniques, and how we overcome these limitations.

2.1 Passage Interactions in Cross-Encoders

Pre-trained transformer-based language models currently are the most effective for passage re-ranking [40]. The models can be divided into two types: cross-encoders and bi-encoders. Cross-encoders receive a query and a passage as input and predict the relevance of the passage [1, 36, 45, 49]. Note that, for brevity, we also refer to other transformer-based language models, be they encoder-only, decoder-only, or encoder-decoders that process a query and a passage simultaneously, as cross-encoders. In contrast to cross-encoders, bi-encoders encode a query and a passage into separate embedding vectors to obtain semantic representations [61], scoring relevance via vector similarity. Bi-encoders are more efficient than cross-encoders since their passage representations can be indexed offline [67]. However, cross-encoders are more effective as they model query–passage interactions.

Current cross-encoders lack a central property that previously boosted the effectiveness of feature-based learning-to-rank models [9, 38, 52, 53, 55]. They cannot model passage interactions and be permutation-invariant at the same time. Passage interactions improve the effectiveness of re-ranking models by allowing passages to exchange information. Permutation invariance is essential for efficiency. In its absence, a model is sensitive to ranking perturbations and one must resort to testing multiple permutations to achieve maximum effectiveness.

Re-ranking models could use corpus-level [6, 44] or ranking-level [51, 57, 83] passage interactions. *Corpus*-level passage interactions provide information about a passage given all other passages in the corpus. For example, PageRank [6] can be used to determine a passage's importance based on the link graph. Or the graph induced by passage similarities in the corpus can be analyzed [44]. However, even if corpus-level passage interactions may be useful for effective retrieval [83], they are difficult to integrate into transformer-based language models [42, 82], and query-independent.

Therefore, we focus on *ranking*-level passage interactions which allow for scoring passage relevance dependent on the query and other passages within the ranking. Previous work has shown that incorporating ranking-level passage interactions can improve the effectiveness of cross-encoders [51, 57, 83]. For example, duo cross-encoders concatenate the query and two passages to predict which passage of the input pair should be ranked higher [51, 57]. However, predictions of duo cross-encoders are not symmetric and not transitive [32]; switching the order of the input passage pair can lead to a different ranking preference. Therefore, duo cross-encoders must score all passage pairs for maximum effectiveness. This problem is alleviated by increasing the number of passages that are simultaneously passed to the transformer model. Recent decoder-only LLMs simultaneously re-rank up to 20 passages [58, 59, 69]. However, running LLMs with billions of parameters is expensive, and many permutations need to be tested for maximum effectiveness, despite heuristics to minimize the number of permutations to test [72].

Existing cross-encoders that model passage interactions are not permutation-invariant because they concatenate passages in the input. Our Set-Encoder follows a different strategy and processes passages in parallel. Like a bi-encoder, the Set-Encoder encodes the semantic information of a passage in a single embedding vector. This embedding vector is shared with all other passages, enabling passage interactions while ensuring permutation invariance. This strategy is similar to the Fusion-in-Encoder (FiE) [37]. FiE enables passage interactions within the encoder by adding additional so-called global tokens. Passage tokens can attend to global tokens and vice-versa. However, no direct interaction between passage

[1] Code and data: https://github.com/webis-de/set-encoder.

tokens is possible. In contrast, our Set-Encoder allows for direct interactions between passage tokens, enabling more effective passage interactions.

2.2 Fine-Tuning Cross-Encoders

The MS MARCO dataset [48] is now the most commonly used dataset for fine-tuning cross-encoders, as it contains many query–passage pairs. The major drawback of this dataset is that most queries have only a single passage labeled as relevant. This label sparsity has two implications: (1) the options for suitable loss functions are limited, and (2) "non-relevant" passages have to be sampled heuristically, leading to noisy data.

Regarding the first implication, initial work resorted to pointwise or pairwise loss functions [49, 50]. More recently, listwise losses have been shown to produce more effective models [31, 56, 70, 84]. Instead of computing the loss using a single relevant and one non-relevant passage, a set of k non-relevant passages is considered. Generally, the more non-relevant passages are included in a training sample, the more effective models are. The highest reported number of passages is $k = 36$ [84], with most work citing resource constraints as the limiting factor for further increasing k. The quadratic memory requirements of the transformer's self-attention mechanism [73] make it difficult to fine-tune models on more passages. We rely on recent work on memory-efficient fused-attention kernels [25, 26, 39] to circumvent the memory constraints and fine-tune models on the full set of up to 100 passages, which they then also receive during re-ranking.

Regarding the second implication, previous work has mostly relied on "hard negative" sampling, i.e., using an effective first-stage retrieval model to sample "non-relevant" samples. The more effective the first-stage retrieval model is, the more difficult it is for the cross-encoder to assess which passage is sampled, and the more effective the fine-tuned model becomes [31]. This sampling technique, however, has its limits. Previous work found that the corpus often contains passages that are actually more relevant than the labeled passage for a substantial number of queries [2]. It was found that models can pick up on biases in the labeling process, overfitting the training data.

To obtain training data of higher quality, Sun et al. [69] propose using LLMs in a zero-shot manner to re-rank passages retrieved by a first-stage retrieval model. Using a permutation distillation loss, a cross-encoder is then fine-tuned to mimic the LLM's ranking [71]. This approach has several advantages over using the MS MARCO labels and heuristic negative samples. First, since the LLM is applied in a zero-shot manner, it is unaffected by the biases of the MS MARCO labeling process. Second, the resulting data is densely labeled; each passage's rank is considered its label. Third, the improved ranking of the LLM reduces noise in the training data. However, distilling cross-encoders from LLMs also comes with drawbacks, namely that the effectiveness of the LLM is the upper bound of the cross-encoder, and that it is unclear if LLMs can reliably produce relevance labels for information retrieval setups [28]. In our case, we do not use an LLM to judge passages but instead use its predictions to rank them (which is much cheaper than human assessors [28]), and then distill it into a cross-encoder based on its rankings.

Sun et al. [69] released the top 20 passages retrieved by BM25, re-ranked by OpenAI's GPT-3.5 for 100k queries. While the scale of the dataset makes it a valuable resource, we find several points for improvement. Chiefly among them, previous insights for improving the fine-tuning cross-encoders on MS MARCO have not yet been applied, namely using a more effective first-stage retrieval model to sample data [31], and second, fine-tuning cross-encoders on as many passages as possible [84]. Rectifying these two issues, we build and release a new dataset for listwise distillation of cross-encoders. We use ColBERTv2 [66] as first-stage retrieval model and re-rank the top 100 passages using OpenAI's GPT-4 Turbo.

3 THE SET-ENCODER MODEL

In this section, we introduce our new cross-encoder architecture: the Set-Encoder. It uses a novel inter-passage attention pattern to model permutation-invariant interactions between passages.

To enable permutation-invariant interactions between multiple passages in a cross-encoder, the three main challenges are: (1) the passages must be able to attend to each other, (2) the interactions between the passages must not encode any positional information about the passage ordering, and (3) the interactions should be "lightweight" enough for efficient cross-encoder training / fine-tuning.

Previous work has addressed the first challenge (attention between passages) by concatenating the query and multiple passages into one input sequence for a transformer-based language model; visualized in Figure 1 (right). Concatenation, however, "violates" the second challenge: the language model's positional encodings (used to determine the token positions) span across passage boundaries and thus encode the order of the passages in the sequence. A different input ordering of the passages can then change the output re-ranking. Furthermore, concatenation also violates the third challenge (efficiency) as the language model's computational cost scales quadratically with the length of the input sequence.

Instead of concatenating passages, we use a novel pattern: inter-passage attention, as visualized in Figure 1 (left). Inter-passage attention processes the passages in parallel with individual input sequences per passage—similar to batched processing of standard cross-encoders. Each of the individual input sequences—one per passage—has its positional encodings starting from zero so that no information about the passage order is encoded (second challenge). To still let the passages in a batch attend to each other (first challenge), we use a special [CLS] token per sequence that aggregates semantic information about its passage and to which tokens from other sequences it can attend. The interactions between passages then are based on just these single [CLS] tokens per passage and not on full-attention between all tokens of different passages. This makes inter-passage attention computationally efficient (third challenge). In principle, our novel pattern also supports passing multiple queries to a model. The inter-passage attention is then only enabled between input sequences with the same query. We formalize the described ideas in the below sections.

3.1 Permutation-Invariant Input Encoding

Standard cross-encoders compute relevance scores for query–passage pairs (q, d) given as token sequences $t_{1_q} \dots t_{m_q}$ and $t_{1_d} \dots t_{m_d}$. Using a BERT-style encoding [27], the final tokens t_{m_q} and t_{m_d} are

special [SEP] separator tokens. Prepended by a special [CLS] classification token c, the concatenated sequence $c\ t_{1_q} \dots t_{m_q}\ t_{1_d} \dots t_{m_d}$ is passed through a transformer-based encoder model, and the relevance score of d for q is computed by a linear transformation on the final contextualized embedding of the [CLS] token c.

The Set-Encoder gets a query q and a set of passages $\{d_1, \dots, d_k\}$ as input and computes a relevance score for each passage. To this end, the Set-Encoder builds a set of input sequences by prepending a [CLS] token and the query sequence to each passage individually. The set of input sequences is then processed simultaneously similar to batched processing with a standard cross-encoder—but with attention from an input sequence's tokens to the class tokens of the other input sequences (see Section 3.2). The batched input sequences are passed through an encoder, and the relevance scores of the passages d_i for q are computed by a linear transformation on the passage's final contextualized class token embedding.

In contrast to concatenating passages, our batched input encoding is permutation-invariant as each input sequence starts its positional encodings from zero. This means that the first token of each passage is at position $m_q + 1$. The model then cannot distinguish between different permutations of the input sequences and, therefore, also not between different permutations of the passages.

3.2 Inter-Passage Attention

In the Set-Encoder, we use a special form of attention to model passage interactions: we only allow every sequence to attend to the [CLS] tokens of every other sequence. Our intuition is that these [CLS] tokens aggregate the semantic information from their sequence and can share it with all other sequences.

Using single tokens for passage interactions is on the efficiency end of a spectrum of possible variants for exchanging information between passages. Sharing more tokens can make the information exchange between passages more fine-grained but definitely makes it computationally more expensive. Increasing the number of passage interactions increases the computational cost quadratically. The extreme case of concatenating passages and thus having every token attend to all other tokens might offer more potential for information exchange but it also is particularly inefficient. We hypothesize that using single tokens for information exchange offer a dramatic improvement of efficiency at sufficient effectiveness.

Our hypothesis is motivated by the semantic information aggregation in many model architectures. For instance, in bi-encoders, the [CLS] tokens capture the semantic information of a passage and in standard cross-encoders, the [CLS] tokens aggregate query–passage information to compute a relevance score. We hypothesize that a [CLS] token then is also able to share this information with other passages and select the [CLS] tokens as the single tokens that ensure passage interactions in the Set-Encoder. The Set-Encoder's novel inter-passage attention pattern thus lets an input sequence of [CLS], query, and passage tokens attend to itself and to all other input sequences' [CLS] tokens.

To implement the novel pattern, we modify the original dot-product attention function of the transformer [73]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{h}}\right)V,$$

where Q, K, and V are matrices of embedding vectors and h is the dimensionality of the embeddings. Each vector in the matrices corresponds to a token from the transformer's input of two token sequences ('origin' and 'target') for which the transformer derives the attention. With x as the length of the origin and y as the length of the target sequence, Q is an $x \times h$ matrix containing the origin sequence's token embedding vectors, and K and V are $y \times h$ matrices containing embedding vectors of the target sequence. Normalizing and softmaxing the $x \times y$ product matrix QK^T yields probabilities of how much the i-th token from the origin sequence attends to the j-th token from the target sequence. The transformer's output $\text{Attention}(Q, K, V)$ then is an $x \times h$ matrix of probability-weighted target sequence embedding vectors from V.

The Set-Encoder uses an encoder transformer model. Encoders employ self-attention (i.e., the origin and target sequence are identical), so that their input is just one sequence from which each token can attend to all tokens from within the sequence. For the Set-Encoder's batched input representation, input sequences are instead embedding vector matrices $Q^{(i)}$, $K^{(i)}$, and $V^{(i)}$ for each passage d_i's input sequence. To allow the i-th input sequence to attend to the [CLS] tokens of all other input sequences, we append the [CLS] token's embedding vectors from the other input sequences' target embedding matrices $K^{(j)}$ and $V^{(j)}$ to the input sequence's target embedding matrices $K^{(i)}$ and $V^{(i)}$. Specifically, let $\bar{K}^{(i)} = [K_1^{(j)} : j \neq i]$ and $\bar{V}^{(i)} = [V_1^{(j)} : j \neq i]$ be the concatenated matrices of the [CLS] token's embedding vectors from K and V of every passage except passage d_i, with $[\cdot\cdot]$ denoting column-wise vector / matrix concatenation (i.e., $[MM']$ is equal to a matrix whose "left" columns come from M and whose "right" columns come from M'). Once the [CLS] tokens from the other passages' embedding vectors have been appended to the target sequence matrices, we obtain the inter-passage attention pattern: $\text{Attention}(Q^{(i)}, [K^{(i)}\bar{K}^{(i)}], [V^{(i)}\bar{V}^{(i)}])$.

4 CROSS-ENCODER TRAINING

The standard paradigm for training cross-encoders is to use passages that have been assessed as relevant from the MS MARCO dataset [48] and sample pseudo non-relevant passages from the retrieved passages of a first-stage retrieval model. Recently, a new paradigm has emerged in which a cross-encoder is distilled to mimic the ranking of a highly effective LLM [69]. Both paradigms have advantages and disadvantages but have only been applied in a complementary manner. We discuss the pros and cons of each paradigm and how we combine them to obtain the advantages from both approaches.

4.1 Sampling Negative Passages

We first discuss the standard paradigm of sampling negatives using a first-stage retrieval model. Its main advantage is that obtaining a sizable training dataset is relatively cheap. The MS MARCO dataset contains over 500k queries with, on average, one relevant passage each. Since first-stage retrieval models are usually very efficient, sampling negative "non-relevant" passages is fast and inexpensive. The main drawbacks are that the training data is noisy and that the single relevant sample per query limits the options for loss functions. The noise stems from the fact that the passage marked

as relevant for a query is usually not the only relevant passage and often not even the most relevant passage [2]. Consequently, sampled negative "non-relevant" passages are often more relevant than those marked as relevant.

Two main strategies to counteract the training disadvantages and fine-tune effective cross-encoders have emerged from the literature. The first strategy is to use a highly effective first-stage ranker which is used to sample "hard negatives". For example, sampling negative passages using a neural retrieval model such as ColBERTv2 yields more effective cross-encoders than using BM25 [31]. The second strategy is to use a listwise loss function with as many negative "non-relevant" samples as possible [56].

Previous work was able to fine-tune cross-encoders with only a limited number of negative samples due to resource constraints. The highest number we are aware of is $k = 36$ [84]. We circumvent this resource constraint by using memory-efficient fused-attention kernels [39]. We aim to investigate whether a cross-encoder or the Set-Encoder profits from fine-tuning on the same number of passages as it will see during re-ranking—up to 100 passages. This is especially important for the Set-Encoder since encoder models are sensitive to shifts between training and inference data [47]. For instance, fine-tuning the Set-Encoder on 8 passages but running inference on 100 passages will likely result in a stronger drop in effectiveness compared to a standard cross-encoder. The passage distribution during inference would substantially differ from the training distribution, which is also known to cause problems in traditional learning-to-rank models [46].

4.2 LLM Distillation

A different and more recent paradigm for fine-tuning cross-encoders is to distill a model from the rankings of a more effective but considerably less efficient decoder-only LLM, as proposed by [69]. The main advantage of this paradigm is that the noise in the training data is reduced, especially on MS MARCO, where relevance judgments of LLMs are highly correlated with human expert judgments [28]. The LLM is applied in a zero-shot setting, i.e., it was not explicitly trained or prompted for MS MARCO queries or documents so that the LLM is not affected by the sparsity of the labels (albeit contamination effects might occur [43], because it is unclear what proportion of MS MARCO the model implicitly saw during training). Additionally, since the LLM assigns a rank to each passage, the training data is densely labeled, allowing for the use of more nuanced loss functions. The main drawback is that the training data is comparatively expensive to obtain. The most effective decoder-only LLMs for re-ranking, referred to as RankGPT, use closed-source GPT models hosted by OpenAI. Inference costs for RankGPT depend on the model but can quickly exceed hundreds of dollars [69]. Open-source alternatives exist but do not match the effectiveness of RankGPT [58, 59].

Sun et al. [69] released a sizable dataset of 100k queries with the top 20 passages as retrieved by BM25 re-ranked by RankGPT-3.5. While the scale of this dataset is impressive, we note several aspects in which it can be improved. Due to the large scale, the weaker (compared to GPT-4 Turbo) GPT-3.5 model was used to save costs. We hypothesize a higher quality but smaller dataset generated with GPT-4 Turbo will yield more effective models. In addition, the dataset does not follow the strategies previous work showed to improve cross-encoder effectiveness. First, BM25 is a relatively weak first-stage retrieval model. We checked several samples in preliminary experiments and often found none of the 20 retrieved passages relevant to the query. Second, only 20 passages were retrieved per query. We hypothesize that a more effective first-stage retrieval model and a greater number of passages per query will improve the effectiveness of the Set-Encoder model and cross-encoders in general.

4.3 Two-Stage Fine-Tuning

We employ a two-stage strategy for fine-tuning cross-encoders to achieve high effectiveness while minimizing costs. Instead of only distilling from LLM ranking data, we first fine-tune a model on a large set of noisy data (MS MARCO negative sampling paradigm) and then continue to fine-tune on a smaller but higher-quality dataset (LLM distillation paradigm). Thus, we create our own high-quality LLM distillation dataset. We use ColBERTv2 as the first-stage retrieval model [66] to retrieve 100 passages for 1,000 randomly sampled MS MARCO training queries, then use RankGPT-4 Turbo to re-rank the passages, costing a total of USD $100.

As RankGPT-4 Turbo has a larger context size, we slightly modify the dataset creation process compared to Sun et al. [69]. The limited context size of LLMs normally necessitates using a sliding window approach to re-rank an adequate number of passages. To re-rank 100 passages, Sun et al. [69] first re-rank the bottom 20 passages, then the 70-th to 90-th passages, and so on. This windowed strategy is expensive as many passages are re-ranked twice. To re-rank 100 passages, 180 passages are passed to the model. The windowed strategy also has a strong positional bias, as a passage ranked in the top 10 by the first stage retrieval model can, in the worst case, be re-ranked to position 20. RankGPT-4 Turbo can re-rank all 100 passages simultaneously. While re-ranking all passages simultaneously is cheaper, it may lead to lower effectiveness. LLMs have implicit positional biases when re-ranking passages [72]. To verify these biases do not substantially influence RankGPT-4 Turbo, we run a pilot study on the TREC Deep Learning 2019 and 2020 tracks [23, 24]. We run RankGPT-4 (using the windowed strategy) and RankGPT-4 Turbo on the top 100 passages retrieved by ColBERTv2. Both models achieve similar effectiveness, with RankGPT-4 and RankGPT-4 Turbo reaching 0.779 and 0.770 nDCG@10, respectively, highlighting that the windowed strategy is not required for achieving high effectiveness and that RankGPT-4 Turbo is a suitable replacement for windowed RankGPT-4.

4.4 Fine-Tuning Settings

We mostly follow Pradeep et al. [56] for fine-tuning baseline models and the Set-Encoder. We use an $\text{ELECTRA}_{\text{BASE}}$ [10] checkpoint from HuggingFace [80] as starting point for fine-tuning (`google/electra-base-discriminator`). For the initial fine-tuning, we use the relevance labels from the MS MARCO passage dataset [48] and sample hard-negatives from the top-200 passages retrieved by ColBERTv2 [66]. We truncate sequences longer than 256 tokens and train monoELECTRA and Set-Encoder models for 50K steps with a batch size of 32 using LCE loss [31] and 7 negative examples per query. Separate monoELECTRA and Set-Encoder models are

then further fine-tuned for 320 steps (10 epochs of our new dataset) on the previously released RankGPT-3.5 distillation dataset and our new RankGPT-4 Turbo distillation dataset using the RankNet loss function [7]. For comparison, we continue fune-tuning a monoELECTRA and Set-Encoder model with 99 hard-negatives using LCE loss. We additionally fine-tune monoELECTRA and Set-Encoder models for 320 steps from scratch using both distillation datasets for comparison. We use the AdamW [41] optimizer with a learning rate of 10^{-5}. All models were trained on a single NVIDIA A100 40GB GPU and implemented using PyTorch [54] and Lightning [29].

5 EVALUATION

To evaluate the effectiveness of our proposed dataset and model, we conduct experiments on the TREC Deep Learning 2019 and 2020 tracks [23, 24] and the TIREx platform [30]. Our analyses are guided by the following research questions, which we investigate in subsequent subsections:

- **RQ1** How does the quality of listwise training data affect the re-ranking effectiveness of cross-encoders?
- **RQ2** Can inter-passage information improve the re-ranking effectiveness of cross-encoders?
- **RQ3** How important is permutation invariance for the robustness and efficiency of cross-encoders?

5.1 LLM Distillation Fine-Tuning

To investigate our first research question we compare the various fine-tuning configurations described in Section 4.4 on the TREC Deep Learning 2019 and 2020 tracks [23, 24]. We measure and report the nDCG@10 of each model re-ranking the top 100 passages retrieved by BM25 [64] and ColBERTv2 [66]. We compare the effectiveness with RankGPT-4 [69], RankGPT-4 Turbo, and LiT5-Distill [71]. LiT5 is a recently proposed cross-encoder based on the T5 model [60]. It is fine-tuned on the RankGPT-3.5 distillation dataset and can model interactions between passages but is not permutation invariant because it encodes the rank position of the passage in the input. It additionally has an explicit position bias as it uses the same sliding window strategy as RankGPT (see Section 4.2). Table 1 shows the results for the various fine-tuning configurations.

We first investigate the effects of the configurations on monoELECTRA. Fine-tuning monoELECTRA using only RankGPT distillation data produces rather ineffective models. They are substantially worse than RankGPT-4 and RankGPT-4 Turbo when re-ranking passages retrieved by BM25. When re-ranking passages retrieved by ColBERTv2, the models are even worse than the first-stage retrieval model. This is in stark contrast to LiT5, which is also only fine-tuned using permutation distillation. LiT5 achieves similar effectiveness on BM25 but is considerably better on ColBERTv2. We attribute this to the explicit positional bias from the windowed strategy. We investigate this effect further in Section 5.3.

Fine-tuning using hard-negative produces more effective models. But, similar to previous work, we find that alignment between the first-stage retrieval model and the training data is important [31]. The model fine-tuned on 8 samples using ColBERTv2 hard-negatives is almost as effective as both RankGPT-4 and RankGPT-4 Turbo when re-ranking passages retrieved by ColBERTv2 but is substantially less effective when re-ranking passages retrieved by BM25.

Table 1: Comparison of nDCG@10 on the TREC Deep Learning 2019 and 2020 [23, 24] tracks of monoELECTRA and Set-Encoder models using either hard-negative sampling (HN) or RankGPT distillation (RG) for initial fine-tuning (1. FT) and subsequent fine-tuning (2. FT) with the number of passages per sample given in parentheses. The highest and second-highest scores per task are bold and underlined, respectively.

Model	1. FT	2. FT	BM25		ColBERTv2	
			2019	2020	2019	2020
First Stage	–	–	0.480	0.494	0.732	0.725
RankGPT-4	–	–	0.713	**0.713**	0.766	**0.793**
RankGPT-4 T.	–	–	0.716	0.699	**0.777**	0.764
LiT5-Distill	RG-3.5 (20)	–	0.696	0.679	0.753	0.744
monoELEC.	HN (8)	–	0.668	0.681	0.760	0.746
		HN (100)	0.677	0.685	0.757	0.739
		RG-3.5 (20)	0.697	0.689	0.757	0.740
		RG-4T (100)	**0.730**	0.710	0.769	0.762
	RG-3.5 (20)	–	0.695	0.645	0.717	0.677
	RG-4T (100)	–	0.681	0.672	0.727	0.677
Set-Encoder	HN (8)	–	0.633	0.645	0.728	0.677
		HN (100)	0.668	0.671	0.751	0.731
		RG-3.5 (20)	0.691	0.668	0.721	0.699
		RG-4T (100)	0.725	0.704	0.770	0.752
	RG-3.5 (20)	–	0.676	0.645	0.708	0.687
	RG-4T (100)	–	0.672	0.659	0.716	0.687

Continuing to fine-tune the model fine-tuned on hard-negatives with 100 instead of 8 samples barely changes the effectiveness. The potential of fine-tuning on hard negatives seems to be exhausted. Continuing to fine-tune on the less noisy RankGPT-3.5 distillation dataset similarly does not change the effectiveness.

In contrast, fine-tuning on our new RankGPT-4 Turbo data improves effectiveness substantially, especially for re-ranking the passages retrieved by BM25. Improvements for ColBERTv2 re-ranking are marginal, but they close the slight effectiveness gap to RankGPT-4 and RankGPT-4 Turbo. On the BM25 passages, monoELECTRA achieves the highest effectiveness on TREC DL 2019 and the second highest on the 2020 track of all models, even outperforming the RankGPT-4 Turbo model that it was distilled from. This effect is caused by the lack of permutation invariance of RankGPT, which we investigate in more detail in Section 5.3.

The effect of the various fine-tuning configurations on the Set-Encoder is much different. Like monoELECTRA, using LLM distillation data as the initial fine-tuning dataset produces poor Set-Encoder models. However, while fine-tuning on 8 hard-negative samples is effective for monoELECTRA, it leads to poor effectiveness for the Set-Encoder. Further fine-tuning the model on the RankGPT-3.5 data with 20 samples leads to a slight improvement, but the model is still substantially less effective than monoELECTRA using the same configuration. Only when the Set-Encoder is fine-tuned on 100 samples can it match monoELECTRA's effectiveness. The model fine-tuned with 100 hard-negative samples and the

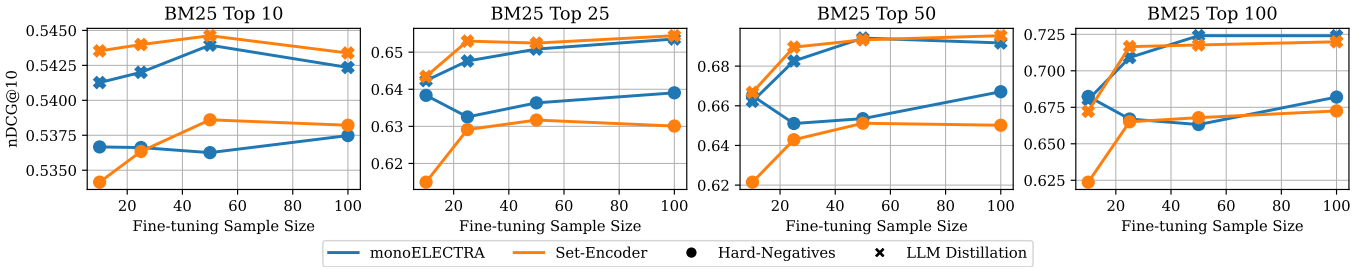

Figure 2: nDCG@10 of monoELECTRA and Set-Encoder models fine-tuned on different numbers of passages $k \in \{10, 25, 50, 100\}$ using hard-negative and LLM distillation data and re-ranking the top k passages retrieved by BM25.

model fine-tuned with our RankGPT-4 Turbo data achieve similar effectiveness as monoELECTRA in its respective configurations.

In summary, when first fine-tuned with hard negatives and then with our distillation dataset, the Set-Encoder is on par with monoELECTRA overall, slightly less effective than both RankGPT models when re-ranking ColBERTv2, and slightly more effective than RankGPT on passages retrieved by BM25. The fact that monoELECTRA and the Set-Encoder are on par suggests that interactions between passages do not improve re-ranking effectiveness. Section 5.2 investigates this in more detail. The need to fine-tune the Set-Encoder on 100 passages suggests it is sensitive to a shift between training and inference data.

To gain a better understanding of the effect, we further fine-tuned additional monoELECTRA and Set-Encoder models for different numbers of passages $k \in \{10, 25, 50, 100\}$ per fine-tuning sample using ColBERTv2 hard-negatives and our RankGPT-4 Turbo data. Figure 2 visualizes the nDCG@10 for each model re-ranking the top k passages retrieved by BM25. Supporting our hypothesis, we observe that the Set-Encoder's effectiveness increases with increasing sample size for both hard-negative and LLM distillation data. The effect is less pronounced when re-ranking fewer passages, i.e., a Set-Encoder model fine-tuned on 100 passages can still re-rank less than 100 passages effectively. To effectively re-rank 100 passages, the Set-Encoder model should be fine-tuned on at least 50 passages.

We also observe an effectiveness gap between the hard-negative training data (circles) and our LLM distillation data (crosses) for monoELECTRA and the Set-Encoder. Our LLM distillation data improves model effectiveness for almost all combinations of re-ranking depth and fine-tuning sample size. The difference is less pronounced at lower sample sizes, especially for monoELECTRA. A larger sample size leads to more effective monoELECTRA models when fine-tuned on our distillation data, while the effectiveness for monoELECTRA fine-tuned on hard-negatives stays more or less constant with increasing sample size. This suggests that the distillation dataset should contain at least 25 or 50 samples for a model to profit from high-quality LLM distillation data.

In conclusion, a two-stage fine-tuning approach, with hard-negative samples first and LLM distillation data second, works well for fine-tuning cross-encoders. Second, LLM distillation data must be high-quality and have an adequate depth to have a noticeable effect compared to hard-negative data. Finally, the Set-Encoder must be fine-tuned with an adequate number of samples to be effective.

Table 2: Overview of the corpora and tasks used for evaluation. For each corpus, we report the mean document length (of the top 100 retrieved by BM25), the number of queries, and the average judgments per query.

Corpus	Tasks		Queries		Documents
	Details	#	Judg.	#	Length
Antique	QA Benchmark [33]	1	32.9	200	49.9
Args.me	Touché [3, 4]	2	60.7	99	435.5
ClueWeb09	Web Tracks [12–15]	4	421.8	200	1132.6
ClueWeb12	Web Tracks, Touché [3, 4, 18, 19]	4	163.8	200	5641.7
CORD-19	TREC-COVID [76, 79]	1	1386.4	50	3647.7
Cranfield	Fully Judged Corpus [17]	1	8.2	225	234.8
Disks4+5	TREC-7/8, Robust04 [74, 75, 77, 78]	3	1367.4	350	749.3
GOV	TREC Web Tracks [20–22]	3	603.9	325	2700.5
GOV2	TREC TB [8, 11, 16]	3	902.3	150	2410.3
MEDLINE	Genom., PM [34, 35, 62, 63]	4	518.3	180	309.1
MS MARCO	TREC DL [23, 24]	2	212.8	97	77.1
NFCorpus	Medical LTR Benchmark [5]	1	48.7	325	364.6
Vaswani	Scientific Abstracts	1	22.4	93	51.3
WaPo	Core '18	1	524.7	50	713.0

5.2 Inter-Passage Attention

Next, we investigate our second research question regarding the effect of inter-passage attention on cross-encoders. We evaluate the Set-Encoder and monoELECTRA models fine-tuned using our RankGPT-4 Turbo distillation data on the TIREx platform [30]. The platform combines 31 TREC-style retrieval tasks across 14 corpora. See Table 2 for an overview. Note that TIREx includes the TREC Deep Learning 2019 and 2020 tracks under the MS MARCO corpus. Since we fine-tune our models on MS MARCO, it serves as the in-domain evaluation corpus. All other corpora are out-of-domain or zero-shot evaluation corpora. We follow Fröbe et al. [30] and re-rank the top 100 passages retrieved by either ChatNoir (for the ClueWeb corpora) or BM25 (for all other corpora). We measure nDCG@10 and micro-average the scores across all queries associated with a corpus. We also report the arithmetic mean and geometric mean across all corpora. Table 3 shows the results for our fine-tuned monoELECTRA and Set-Encoder models, various sizes of monoBERT [49] and monoT5 [50], a sparse cross-encoder model [68], and LiT5-Distill [71].

Table 3: Comparison of various cross-encoder and LLM re-ranking models in terms of nDCG@10 micro-averaged across all queries from a corpus from the TIREx benchmark [30]. Macro-averaged arithmetic and geometric means are computed across all corpora. Model sizes are given in the number of parameters. The highest and second-highest scores per corpus are bold and underlined, respectively.

Model	Parameters	Antique	Args.me	ClueWeb09	ClueWeb12	CORD-19	Cranfield	Disks4+5	GOV	GOV2	MEDLINE	MS MARCO	NFCorpus	Vaswani	WaPo	A. Mean	G. Mean
First Stage	–	0.516	0.404	0.178	**0.364**	0.586	0.012	0.436	0.237	0.466	0.358	0.487	0.281	0.447	0.364	0.367	0.394
Sparse monoMiniLM	11M	0.545	0.312	0.198	0.312	0.673	**0.014**	0.498	0.267	0.502	0.228	0.691	0.299	0.436	0.434	0.386	0.427
monoBERT base	110M	0.512	0.314	0.192	0.263	**0.690**	0.009	0.531	0.265	0.489	0.256	0.701	0.310	0.321	0.449	0.379	0.422
monoBERT large	340M	0.489	0.371	0.134	0.251	0.625	0.009	0.504	0.250	0.474	0.296	0.714	0.303	0.476	0.438	0.381	0.422
monoT5 base	220M	0.510	0.304	0.186	0.260	0.688	0.009	0.535	0.267	0.486	0.253	0.705	0.310	0.306	0.451	0.376	0.420
monoT5 large	770M	0.532	0.337	0.182	0.266	0.636	0.010	0.566	0.267	0.512	0.313	0.717	0.311	0.414	**0.492**	0.397	0.438
monoT5 3b	3B	0.543	0.391	0.201	0.279	0.603	0.011	**0.569**	**0.292**	0.513	0.348	**0.736**	**0.324**	0.458	0.476	**0.410**	**0.448**
LiT5-Distill	220M	**0.576**	0.394	**0.215**	0.275	0.686	0.011	0.509	0.268	**0.534**	0.334	0.687	0.293	0.429	0.470	0.406	0.445
monoELECTRA	110M	0.539	0.379	0.174	0.288	0.674	0.008	0.460	0.229	0.513	0.366	0.720	0.280	0.482	0.435	0.396	0.438
Set-Encoder	110M	0.569	**0.406**	0.175	0.296	0.663	0.007	0.474	0.253	0.516	**0.369**	0.715	0.287	**0.507**	0.443	0.406	0.446

Compared to the other models, the Set-Encoder reaches the second-highest effectiveness on average across all corpora for both arithmetic and geometric mean. Only the largest monoT5 model is marginally more effective but uses 27-times more parameters. Despite being less effective for in-domain retrieval, LiT5-Distill is also similarly effective on average across all corpora. However, it uses twice the number of parameters and is not permutation invariant, which we investigate further in Section 5.3. Additionally, our fine-tuned monoELECTRA is on par with monoT5 large, the previously second-best model on TIREx, highlighting the effectiveness of our two-stage fine-tuning approach.

In the direct comparison to monoELECTRA, for in-domain retrieval on MS MARCO, the Set-Encoder model is marginally less effective. However, the difference is not significant (paired t-test, $p < 0.01$ Bonferroni-corrected). For out-of-domain retrieval, the Set-Encoder improves over monoELECTRA on 11 out of 13 corpora. We measure significant improvements on the Antique, Args.me, and Gov corpora. Across all corpora, the Set-Encoder is more effective regarding both arithmetic and geometric mean.

To better understand how the Set-Encoder uses inter-passage attention, Figure 3 visualizes the attention saliency per token type of monoELECTRA and the Set-Encoder. We average the attention probabilities of the [CLS] token to itself, query tokens, passage tokens, and other passages' [CLS] tokens per layer for the TREC Deep Learning tracks. In monoELECTRA, the [CLS] token spreads its attention mostly evenly across the token types for the initial layers 1 to 8. In layers 9 through 10, the model focuses almost exclusively on the passage tokens. In the final 12-th layer, the model aggregates information about the query-passage interaction by balancing attention between the two.

The Set-Encoder's attention pattern differs considerably. In the first two layers, the [CLS] token puts around 18% of its attention on the passage and most of the remaining attention on the other passage's [CLS] tokens. In the layers 3 through 8, the [CLS] token attends almost exclusively to the other passage's [CLS] tokens. This suggests the model aggregates information about the passage in the first two layers and then shares this information with the other passages in the intermediate layers. Like monoELECTRA, in the

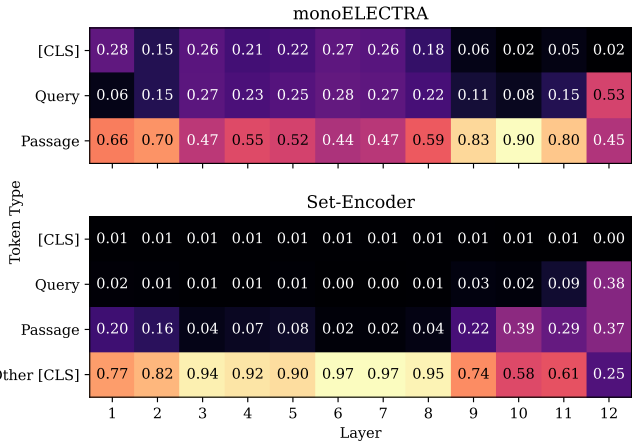

Figure 3: Attention saliency of monoELECTRA and the Set-Encoder of the [CLS] token to itself, query tokens, passage tokens, and other passages' [CLS] tokens per layer.

layers 9 through 11, the [CLS] token increases attention to the passage. The query–passage interaction is then modeled in the final layer, where the [CLS] token also attends to the query tokens.

In conclusion, our new inter-passage attention pattern improves the effectiveness of cross-encoders, especially for out-of-domain re-ranking. It does so by aggregating semantic information about the passage in the initial layers of the encoder model. In the intermediate layers, the semantic information is shared with other passages. The final two layers are used to predict the passage's relevance to the query, incorporating the semantic information of the other passages in the ranking for the query.

5.3 Permutation Invariance

Finally, we investigate our final research question regarding the permutation-invariance of cross-encoder models. We generate several "corrupted" perturbations of the top 100 passages retrieved by BM25 on the TREC Deep Learning 2019 and 2020 tracks: a random

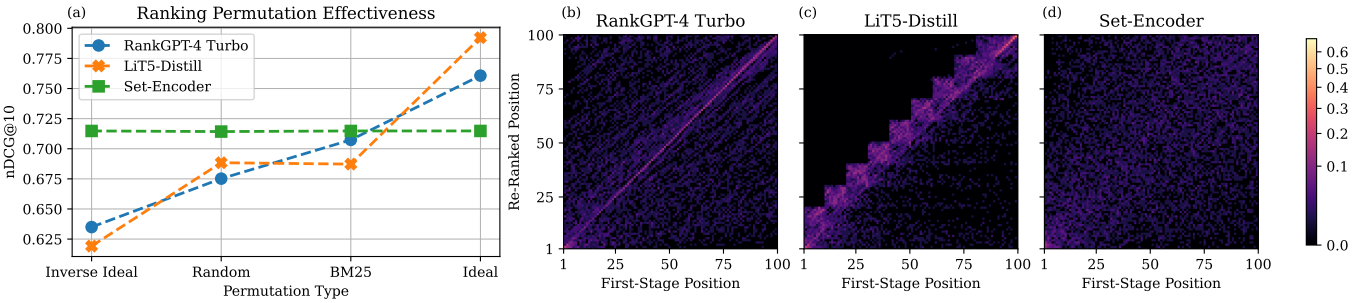

Figure 4: Effectiveness in terms of nDCG@10 for different permutations of a BM25 ranking (a) and proportional rank changes for re-ranking BM25 (b-c) averaged across all queries from the TREC Deep Learning 2019 and 2020 tracks [23, 24].

permutation, an ideal permutation, and a reverse-ideal permutation. Ideal and reverse-ideal permutations are generated by reordering the passages according to their relevance judgments. The random permutation is generated by randomly shuffling the passages. We then measure nDCG@10 of our Set-Encoder model and compare it with RankGPT-4 Turbo and LiT5-Distill. Figure 4 (a) visualizes the results. We additionally investigate the re-ranking behavior of the models in terms of their proportional rank changes on the TREC Deep Learning tasks. Figures 4 (c-d) visualize the proportion of how often a passage at a certain position in the BM25 first-stage ranking is re-ranked by a model to another position.

We observe that the Set-Encoder is robust to ranking perturbations (see Figure 4). It has a constant nDCG@10 of 0.715, irrespective of the ranking permutation. In contrast, RankGPT-4 Turbo and LiT5-Distill are sensitive to the ranking permutation. They are both substantially worse for the reverse-ideal initial ranking, with nDCG@10 of 0.634 and 0.619, respectively. The effectiveness of both models increases with an increasingly better first-stage ranking. The effectiveness of the models on the random permutation is better than on the reverse-ideal permutation but worse than on the unaltered BM25 ranking. Despite increasing effectiveness, RankGPT-4 Turbo and LiT5-Distill are worse than our Set-Encoder model in all those cases. Only when re-ranking with the ideal ranking do the models achieve higher effectiveness than Set-Encoder. This result aligns with our effectiveness comparison in Section 5.1. RankGPT-4 Turbo and LiT5-Distill are on par with the Set-Encoder when re-ranking passages retrieved by the more ColBERTv2 but not when re-ranking BM25.

Analyzing the re-ranking behavior of the models gives insight into why RankGPT-4 Turbo and LiT5-Distill achieve worse effectiveness. Figure 4 (b) visualizes the re-ranking behavior of RankGPT-4 Turbo. It reveals a distinct diagonal band, suggesting the model is biased to maintain a passage's position. This result aligns with previous work finding implicit biases in ranking LLMs [72]. The visualization of the re-ranking behavior of LiT5-Distill in Figure 4 (c) displays, next to a slight diagonal band, a distinct step pattern. This reflects the explicit position bias caused by the windowed strategy (see Section 4.2). Figure 4 (d) visualizes the Set-Encoder's re-ranking behavior. The distribution is more uniform because the model lacks information about a passage's position in the initial ranking.

In conclusion, the Set-Encoder is robust to ranking perturbations because it models interactions between passages in a permutation-invariant way. Models that lack permutation invariance need an already effective ranking to be effective. This can be achieved by, for example, running inference multiple times [72]. This approach is substantially less efficient than using a permutation invariant model. Re-ranking 100 passages with RankGPT-4 Turbo takes about 30 seconds. Using an NVIDIA A100 40 GPU, LiT5-Distill takes about 4 seconds, while our Set-Encoder takes only around 0.3 seconds, and the result is always the same, irrespective of the initial ranking.

6 CONCLUSION

In this paper, we introduced the Set-Encoder, a new cross-encoder that is both permutation-invariant and models interactions between passages. We also built a new high-quality dataset for fine-tuning cross-encoders using ranking distillation from LLMs that densely labels the top 100 passages for 1000 queries. Using our new dataset, we show that effective cross-encoders can be trained by fine-tuning them on a large but noisy dataset and then fine-tuning the model on a smaller set of high-quality data. This training strategy is especially effective for our new Set-Encoder because it can model interactions between more passages than previous models.

Empirical results show that the Set-Encoder is more effective than previous cross-encoders with a similar number of parameters. Compared to larger models, the Set-Encoder is often also more effective. Only the largest and previously most effective models can achieve slightly higher effectiveness. A direct comparison to the same model architecture without inter-passage attention shows that interaction between passages greatly improves effectiveness, especially for out-of-domain scenarios. We additionally find permutation-invariance to be a vital property for cross-encoders. Our experiments show that models that have access to the rank position of the previous stage depend strongly on the effectiveness of the previous retrieval stage. They often benefit from multiple re-ranking cascades that employ a sequence of increasingly expensive models to improve their effectiveness. However, as the Set-Encoder is permutation invariant, such re-ranking cascades can be skipped.

Our proposed model opens many options for future work. One direction is to investigate exchanging or scaling the backbone encoder model. Increasing the model size or using a different encoder architecture could improve effectiveness. Scaling in the opposite direction and using distilled or sparse models could produce highly

effective and very efficient models. Further, LLM distillation allows for more sophisticated listwise loss functions that directly approximate a desired ranking measure. Lastly, as positional information is not provided to the Set-Encoder, there is no dependency between the rank positions of documents provided by the previous stage ranker and those computed by the Set-Encoder. Therefore, another study could investigate comparing the robustness of the Set-Encoder to non-permutation invariant models like RankGPT or LiT5 when the first-stage retrieval system changes between training and test setups.

REFERENCES

[1] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-Domain Modeling of Sentence-Level Evidence for Document Retrieval. In *Proceedings of EMNLP-IJCNLP 2019*. Association for Computational Linguistics, Hong Kong, China, 3490–3496. https://doi.org/10.18653/v1/D19-1352

[2] Negar Arabzadeh, Alexandra Vtyurina, Xinyi Yan, and Charles L. A. Clarke. 2022. Shallow Pooling for Sparse Labels. *Information Retrieval Journal* 25 (2022), 365–385. https://doi.org/10.1007/s10791-022-09411-0

[3] Alexander Bondarenko, Maik Fröbe, Johannes Kiesel, Shahbaz Syed, Timon Gurcke, Meriem Beloucif, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2022. Overview of Touché 2022: Argument Retrieval. In *Proceedings of CLEF 2022 (Lecture Notes in Computer Science, Vol. 13390)*. Springer International Publishing, Bologna, Italy, 311–336. https://doi.org/10.1007/978-3-031-13643-6_21

[4] Alexander Bondarenko, Lukas Gienapp, Maik Fröbe, Meriem Beloucif, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2021. Overview of Touché 2021: Argument Retrieval. In *Proceedings of CLEF 2021 (Lecture Notes in Computer Science, Vol. 12880)*. Springer International Publishing, Virtual Event, 450–467. https://doi.org/10.1007/978-3-030-85251-1_28

[5] Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. 2016. A Full-Text Learning to Rank Dataset for Medical Information Retrieval. In *Proceedings of ECIR 2016 (Lecture Notes in Computer Science, Vol. 9626)*. Springer International Publishing, Padua, Italy, 716–722. https://doi.org/10.1007/978-3-319-30671-1_58

[6] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* 30, 1 (April 1998), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X

[7] Christopher J C Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report MSR-TR-2010-82. Microsoft Research, Redmond, WA. 19 pages. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/MSR-TR-2010-82.pdf

[8] Stefan Büttcher, Charles L. A. Clarke, and Ian Soboroff. 2006. The TREC 2006 Terabyte Track. In *Proceedings of TREC 2006 (NIST Special Publication, Vol. 500–272)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 14. http://trec.nist.gov/pubs/trec15/papers/TERA06.OVERVIEW.pdf

[9] Maarten Buyl, Paul Missault, and Pierre-Antoine Sondag. 2023. RankFormer: Listwise Learning-to-Rank Using Listwide Labels.. In *Proceedings of KDD 2023*. Association for Computational Linguistics, Long Beach, CA, USA, 3762–3773. https://doi.org/10.1145/3580305.3599892

[10] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.. In *Proceedings of ICLR 2020*. OpenReview.net, Addis Ababa, Ethiopia, 14.

[11] Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. 2004. Overview of the TREC 2004 Terabyte Track.. In *Proceedings of TREC 2004 (NIST Special Publication, Vol. 500–261)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 9. http://trec.nist.gov/pubs/trec13/papers/TERA.OVERVIEW.pdf

[12] Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. 2009. Overview of the TREC 2009 Web Track.. In *Proceedings of TREC 2009 (NIST Special Publication, Vol. 500–278)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 9. http://trec.nist.gov/pubs/trec18/papers/WEB09.OVERVIEW.pdf

[13] Charles L. A. Clarke, Nick Craswell, Ian Soboroff, and Gordon V. Cormack. 2010. Overview of the TREC 2010 Web Track.. In *Proceedings of TREC 2010 (NIST Special Publication, Vol. 500–294)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 9. https://trec.nist.gov/pubs/trec19/papers/WEB.OVERVIEW.pdf

[14] Charles L. A. Clarke, Nick Craswell, Ian Soboroff, and Ellen M. Voorhees. 2011. Overview of the TREC 2011 Web Track.. In *Proceedings of TREC 2011 (NIST Special Publication, Vol. 500–296)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 9. http://trec.nist.gov/pubs/trec20/papers/WEB.OVERVIEW.pdf

[15] Charles L. A. Clarke, Nick Craswell, and Ellen M. Voorhees. 2012. Overview of the TREC 2012 Web Track.. In *Proceedings of TREC 2012 (NIST Special Publication, Vol. 500–298)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 8. http://trec.nist.gov/pubs/trec21/papers/WEB12.overview.pdf

[16] Charles L. A. Clarke, Falk Scholer, and Ian Soboroff. 2005. The TREC 2005 Terabyte Track.. In *Proceedings of TREC 2005 (NIST Special Publication, Vol. 500–272)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA,, 11. http://trec.nist.gov/pubs/trec14/papers/TERABYTE.OVERVIEW.pdf

[17] Cyril W. Cleverdon. 1991. The Significance of the Cranfield Tests on Index Languages.. In *Proceedings of SIGIR 1991*. Association for Computing Machinery, Chicago, Illinois, USA, 3–12. https://doi.org/10.1145/122860.122861

[18] Kevyn Collins-Thompson, Paul N. Bennett, Fernando Diaz, Charlie Clarke, and Ellen M. Voorhees. 2013. TREC 2013 Web Track Overview. In *Proceedings of TREC 2013 (NIST Special Publication, Vol. 500-302)*.

[19] Kevyn Collins-Thompson, Craig Macdonald, Paul N. Bennett, Fernando Diaz, and Ellen M. Voorhees. 2014. TREC 2014 Web Track Overview. In *Proceedings of TREC 2014 (NIST Special Publication, Vol. 500-308)*.

[20] Nick Craswell and David Hawking. 2002. Overview of the TREC-2002 Web Track.. In *Proceedings of TREC 2002 (NIST Special Publication, Vol. 500–251)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 10. http://trec.nist.gov/pubs/trec11/papers/WEB.OVER.pdf

[21] Nick Craswell and David Hawking. 2004. Overview of the TREC 2004 Web Track.. In *Proceedings of TREC 2004 (NIST Special Publication, Vol. 500–261)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 9. http://trec.nist.gov/pubs/trec13/papers/WEB.OVERVIEW.pdf

[22] Nick Craswell, David Hawking, Ross Wilkinson, and Mingfang Wu. 2003. Overview of the TREC 2003 Web Track.. In *Proceedings of TREC 2003 (NIST Special Publication, Vol. 500–255)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 78–92. http://trec.nist.gov/pubs/trec12/papers/WEB.OVERVIEW.pdf

[23] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *Proceedings of TREC 2020 (NIST Special Publication, Vol. 1266)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 13. https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf

[24] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2019. Overview of the TREC 2019 Deep Learning Track. In *Proceedings of TREC 2019 (NIST Special Publication, Vol. 500–331)*. National Institute of Standards and Technology, Gaithersburg, Maryland, USA, 22. https://trec.nist.gov/pubs/trec28/papers/OVERVIEW.DL.pdf

[25] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv. https://doi.org/10.48550/arXiv.2307.08691

[26] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Proceedings of NeurIPS 2022*. 16344–16359.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT 2019*. 4171–4186. https://doi.org/10.18653/v1/N19-1423

[28] Guglielmo Faggioli, Laura Dietz, Charles L. A. Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, and Henning Wachsmuth. 2023. Perspectives on Large Language Models for Relevance Judgment. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR 2023)*, Masaharu Yoshioka, Julia Kiseleva, and Mohammad Aliannejadi (Eds.). ACM, Taipei, Taiwan, 39–50. https://dl.acm.org/doi/10.1145/3578337.3605136

[29] William Falcon and The PyTorch Lightning team. 2023. PyTorch Lightning. https://doi.org/10.5281/zenodo.7859091

[30] Maik Fröbe, Jan Heinrich Reimer, Sean MacAvaney, Niklas Deckers, Simon Reich, Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. 2023. The Information Retrieval Experiment Platform. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Taipei Taiwan, 2826–2836. https://doi.org/10.1145/3539618.3591888

[31] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. Rethink Training of BERT Rerankers in Multi-stage Retrieval Pipeline. In *Proceedings of ECIR 2021*. 280–286. https://doi.org/10.1007/978-3-030-72240-1_26

[32] Lukas Gienapp, Maik Fröbe, Matthias Hagen, and Martin Potthast. 2022. Sparse Pairwise Re-ranking with Pre-trained Transformers. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR '22)*. New York, NY, USA, 72–80. https://doi.org/10.1145/3539813.3545140

[33] Helia Hashemi, Mohammad Aliannejadi, Hamed Zamani, and W. Bruce Croft. 2020. ANTIQUE: A Non-factoid Question Answering Benchmark. In *Proceedings of ECIR 2020*. 166–173.

[34] William R. Hersh, Ravi Teja Bhupatiraju, L. Ross, Aaron M. Cohen, Dale Kraemer, and Phoebe Johnson. 2004. TREC 2004 Genomics Track Overview. In *Proceedings of TREC 2004 (NIST Special Publication, Vol. 500-261)*.

[35] William R. Hersh, Aaron M. Cohen, Jianji Yang, Ravi Teja Bhupatiraju, Phoebe M. Roberts, and Marti A. Hearst. 2005. TREC 2005 Genomics Track Overview. In *Proceedings of TREC 2005 (NIST Special Publication, Vol. 500-266)*.

[36] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2021. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. https://doi.org/10.48550/arXiv.2010.02666 arXiv:2010.02666

[37] Akhil Kedia, Mohd Abbas Zaidi, and Haejun Lee. 2022. FiE: Building a Global Probability Space by Leveraging Early Fusion in Encoder for Open-Domain Question Answering. In *Proceedings of EMNLP 2022*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 4246–4260. https://doi.org/10.18653/v1/2022.emnlp-main.285

[38] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 3744–3753.

[39] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, and Daniel Haziza. 2022. xFormers: A modular and hackable Transformer modelling library. https://github.com/facebookresearch/xformers.

[40] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2022. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Springer International Publishing. https://doi.org/10.1007/978-3-031-02181-7

[41] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *Proceedings of ICLR 2019*.

[42] Zhengyi Ma, Zhicheng Dou, Wei Xu, Xinyu Zhang, Hao Jiang, Zhao Cao, and Ji-Rong Wen. 2021. Pre-Training for Ad-hoc Retrieval: Hyperlink Is Also You Need. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. ACM, Virtual Event Queensland Australia, 1212–1221. https://doi.org/10.1145/3459637.3482286

[43] Sean MacAvaney and Luca Soldaini. 2023. One-Shot Labeling for Automatic Relevance Estimation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, Hsin-Hsi Chen, Wei-Jou (Edward) Duh, Hen-Hsen Huang, Makoto P. Kato, Josiane Mothe, and Barbara Poblete (Eds.). ACM, New York, 2230–2235. https://doi.org/10.1145/3539618.3592032

[44] Sean MacAvaney, Nicola Tonellotto, and Craig Macdonald. 2022. Adaptive Re-Ranking with a Corpus Graph. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 1491–1500. https://doi.org/10.1145/3511808.3557231

[45] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. New York, NY, USA, 1101–1104. https://doi.org/10.1145/3331184.3331317

[46] Tom Minka and Stephen Robertson. 2008. Selection bias in the LETOR datasets. In *Proceedings of SIGIR 2008 workshop on learning to rank for information retrieval*, Vol. 2.

[47] Iurii Mokrii, Leonid Boytsov, and Pavel Braslavski. 2021. A Systematic Evaluation of Transfer Learning and Pseudo-labeling with BERT-based Ranking Models. In *Proceedings of SIGIR 2021*. Association for Computing Machinery, New York, NY, USA, 2081–2085. https://doi.org/10.1145/3404835.3463093

[48] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proceedings of COCO@NeurIPS 2016*.

[49] Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage Re-ranking with BERT. https://doi.org/10.48550/arXiv.1901.04085 arXiv:1901.04085

[50] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online, 708–718. https://doi.org/10.18653/v1/2020.findings-emnlp.63

[51] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-Stage Document Ranking with BERT. https://doi.org/10.48550/arXiv.1910.14424 arXiv:1910.14424

[52] Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Virtual Event China, 499–508. https://doi.org/10.1145/3397271.3401104

[53] Rama Kumar Pasumarthi, Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2020. Permutation Equivariant Document Interaction Network for Neural Learning to Rank. In *Proceedings of ICTIR 2020*. 145–148. https://doi.org/10.1145/3409256.3409819

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of NeurIPS 2019*, Vol. 32.

[55] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2020. Context-Aware Learning to Rank with Self-Attention. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR eCom'20)*. https://sigir-ecom.github.io/ecom2020/ecom20Papers/paper18.pdf

[56] Ronak Pradeep, Yuqi Liu, Xinyu Zhang, Yilin Li, Andrew Yates, and Jimmy Lin. 2022. Squeezing Water from a Stone: A Bag of Tricks for Further Improving Cross-Encoder Effectiveness for Reranking.. In *Proceedings of ECIR 2022*. 655–670. https://doi.org/10.1007/978-3-030-99736-6_44

[57] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. arXiv:2101.05667 http://arxiv.org/abs/2101.05667

[58] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. arXiv. https://doi.org/10.48550/arXiv.2309.15088

[59] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking Is a Breeze! arXiv. https://doi.org/10.48550/arXiv.2312.02724

[60] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21 (2020), 140:1–140:67.

[61] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In *Proceedings of EMNLP-IJCNLP 2019*. 3980–3990. https://doi.org/10.18653/v1/D19-1410

[62] Kirk Roberts, Dina Demner-Fushman, Ellen M. Voorhees, William R. Hersh, Steven Bedrick, and Alexander J. Lazar. 2018. Overview of the TREC 2018 Precision Medicine Track. In *Proceedings of TREC 2018 (NIST Special Publication, Vol. 500-331)*.

[63] Kirk Roberts, Dina Demner-Fushman, Ellen M. Voorhees, William R. Hersh, Steven Bedrick, Alexander J. Lazar, and Shubham Pant. 2017. Overview of the TREC 2017 Precision Medicine Track. In *Proceedings of TREC 2017 (NIST Special Publication, Vol. 500-324)*.

[64] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994 (NIST Special Publication, Vol. 500–225)*. 109–126. http://trec.nist.gov/pubs/trec3/papers/city.ps.gz

[65] Guilherme Rosa, Luiz Bonifacio, Vitor Jeronymo, Hugo Abonizio, Marzieh Fadaee, Roberto Lotufo, and Rodrigo Nogueira. 2022. In Defense of Cross-Encoders for Zero-Shot Retrieval. https://doi.org/10.48550/arXiv.2212.06121 arXiv:2212.06121

[66] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. https://doi.org/10.48550/arXiv.2112.01488 arXiv:2112.01488

[67] Harrisen Scells, Shengyao Zhuang, and Guido Zuccon. 2022. Reduce, Reuse, Recycle: Green Information Retrieval Research. In *Proceedings of SIGIR 2022*. 2825–2837. https://doi.org/10.1145/3477495.3531766

[68] Ferdinand Schlatt, Maik Fröbe, and Matthias Hagen. 2023. Investigating the Effects of Sparse Attention on Cross-Encoders. arXiv. https://doi.org/10.48550/arXiv.2312.17649

[69] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In *Proceedings of EMNLP 2023*. Association for Computational Linguistics, 14918–14937. https://doi.org/10.18653/v1/2023.emnlp-main.923

[70] Xingwu Sun, Hongyin Tang, Fuzheng Zhang, Yanling Cui, Beihong Jin, and Zhongyuan Wang. 2020. TABLE: A Task-Adaptive BERT-based ListwisE Ranking Model for Document Retrieval. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM '20)*. New York, NY, USA, 2233–2236. https://doi.org/10.1145/3340531.3412071

[71] Manveer Singh Tamber, Ronak Pradeep, and Jimmy Lin. 2023. Scaling Down, LiTting Up: Efficient Zero-Shot Listwise Reranking with Seq2seq Encoder-Decoder Models. arXiv. https://doi.org/10.48550/arXiv.2312.16098

[72] Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. 2023. Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models. arXiv. https://doi.org/10.48550/arXiv.2310.07712

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Long Beach, CA, 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[74] Ellen Voorhees. 2004. Overview of the TREC 2004 Robust Retrieval Track. In *TREC*.

[75] Ellen M. Voorhees. 1996. NIST TREC Disks 4 and 5: Retrieval Test Collections Document Set.

[76] Ellen M. Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. 2020.

TREC-COVID: constructing a pandemic information retrieval test collection. *SIGIR Forum* 54, 1 (2020), 1:1–1:12.

[77] Ellen M. Voorhees and Donna Harman. 1998. Overview of the Seventh Text Retrieval Conference (TREC-7). In *TREC*.

[78] Ellen M. Voorhees and Donna Harman. 1999. Overview of the Eight Text Retrieval Conference (TREC-8). In *TREC*.

[79] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Chris Wilhelm, Boya Xie, Douglas Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. CORD-19: The Covid-19 Open Research Dataset. arXiv. https://doi.org/10.48550/arXiv.2004.10706

[80] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv. https://doi.org/10.48550/arXiv.1910.03771

[81] 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 *Proceedings of ICLR 2021*. https://openreview.net/forum?id=zeFrfgyZln

[82] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2022. LinkBERT: Pretraining Language Models with Document Links. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 8003–8016. https://doi.org/10.18653/v1/2022.acl-long.551

[83] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, Jean-Marc Langlois, and Yi Chang. 2016. Ranking Relevance in Yahoo Search. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 323–332. https://doi.org/10.1145/2939672.2939677

[84] Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2022. RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses. https://doi.org/10.48550/arXiv.2210.10634 arXiv:2210.10634