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

Ferdinand Schlatt[1], Maik Fröbe[1], Harrisen Scells[2,6], Shengyao Zhuang[3,4], Bevan Koopman[3], Guido Zuccon[4], Benno Stein[5], Martin Potthast[2,6,7], and Matthias Hagen[1]

[1] Friedrich-Schiller-Universität Jena [2] University of Kassel [3] CSIRO
[4] University of Queensland [5] Bauhaus-Universität Weimar
[6] hessian.AI [7] ScadDS.AI
ferdinand.schlatt@uni-jena.de

Abstract. Existing cross-encoder models can be categorized as pointwise, pairwise, or listwise. Pairwise and listwise models allow passage interactions, which typically makes them more effective than pointwise models but less efficient and less robust to input passage order permutations. To enable efficient permutation-invariant passage interactions during re-ranking, we propose a new cross-encoder architecture with inter-passage attention: the Set-Encoder. In experiments on TREC Deep Learning and TIREx, the Set-Encoder is as effective as state-of-the-art listwise models while being more efficient and invariant to input passage order permutations. Compared to pointwise models, the Set-Encoder is particularly more effective when considering inter-passage information, such as novelty, and retains its advantageous properties compared to other listwise models. Our code is publicly available at https://github.com/webis-de/ECIR-25.

1 Introduction

Existing cross-encoders for passage re-ranking [41–43, 50–53, 61, 62, 73] lack a central desirable property of learning-to-rank models [45]: permutation-invariant passage interactions. Passage interactions help to improve effectiveness through information exchange, while permutation invariance ensures that the same ranking is output independent of the ordering of the input passages.

Pointwise cross-encoders process passages independently of each other [41, 42, 50, 73] and thus are permutation-invariant by design but cannot model passage interactions. Pairwise [43, 51] and listwise cross-encoders [52, 53, 61, 62] enable passage interactions by concatenating the passages as a single input sequence. This way, pair- or listwise re-rankers are often more effective than pointwise ones but the derived rankings depend on the concatenation order. To avoid inconsistent rankings for different input orderings, many (or all) permutations are re-ranked and fused to a final ranking; visualized in Figure 1 (bottom).

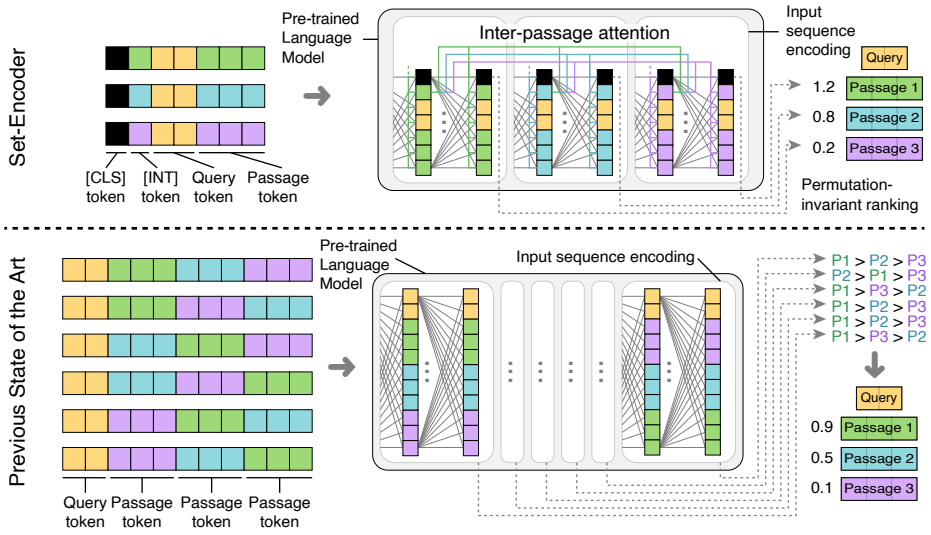

Fig. 1: Comparison of our Set-Encoder architecture (top) with state-of-the-art listwise re-rankers (bottom) for three input passages. List-wise re-rankers concatenate the input passages, leading to potentially inconsistent rankings for different concatenation orderings. Many (or all) permutations are thus re-ranked for optimization. Instead, the Set-Encoder uses novel [INT] tokens for permutation-invariant inter-passage attention.

To directly model passage interactions in a permutation-invariant way, we propose a new cross-encoder architecture: the *Set-Encoder*; visualized in Figure 1 (top). Instead of concatenating the input passages, the Set-Encoder processes them like a pointwise cross-encoder as different sequences in a batch. Passage interactions are enabled by letting each sequence aggregate information in special [INT] interaction embedding tokens that all other sequences can attend to. The [INT] tokens can be interpreted as lightweight bridges which share information between passages. To ensure permutation-invariance, the Set-Encoder assigns every [INT] token the same positional encoding. We call this new attention pattern *inter-passage attention*.

In experiments on the TREC 2019 and 2020 Deep Learning tracks [21, 22] and the TIREx framework [25], the Set-Encoder is as effective as state-of-the-art re-rankers [53, 61, 73]. At the same time, the Set-Encoder is robust to input permutations and orders of magnitude more efficient. Interestingly, contrasting previous work [42, 43, 51, 53, 62, 73], we find listwise interactions to not really be necessary in the Deep Learning and TIREx scenarios. A pointwise model fine-tuned with the currently best available data [60] is as effective as state-of-the-art listwise models and the Set-Encoder. To still demonstrate the potential advantages of listwise models, we fine-tune the Set-Encoder to re-rank passages according to both relevance and novelty and find it to then be more effective than its pointwise counterpart and other state-of-the-art listwise models.

2 Related Work

Pre-trained transformer-based models can roughly be divided into two types [35]: bi-encoders and cross-encoders. Bi-encoders embed queries and passages independently so that pre-computed passage representations can be indexed. At query time, only the query needs to be embedded and can then efficiently be compared with the indexed passage embeddings [54]. Cross-encoders instead embed queries and passages together, allowing for interactions [41]—for brevity, we refer to all such transformer-based models as cross-encoders, be they encoder-only [41], decoder-only [52, 53, 73], or encoder–decoders [42, 73].

This direct query–passage interaction makes cross-encoders particularly effective. However, current pairwise and listwise cross-encoders lack a central property that previously improved feature-based learning-to-rank models [6, 34, 45, 46, 49]: input permutation-invariant interactions. The rank order of current list and pairwise cross-encoders is sensitive to the input passage ordering and often fuse multiple rankings for different input permutations to be effective.

For example, duo cross-encoders embed a query together with two passages to predict which passage of the pair should be ranked higher [43, 51]. But the predictions of duo cross-encoders are neither symmetric nor transitive [26], so that they must process all pairs and twice for maximum effectiveness. More recent decoder-only LLMs that simultaneously re-rank up to 20 passages [52, 53, 61] are also sensitive to the input ordering, so that they usually fuse the re-rankings for several input permutations for maximum effectiveness [63].

Existing cross-encoders that model passage interactions are not permutation-invariant because they concatenate passages into a single input sequence. Our Set-Encoder follows a different strategy and processes passages in parallel. To enable passage interactions while ensuring permutation invariance, we add a new dedicated interaction token ([INT]) to each input sequence. The Set-Encoder allows all passages to attend to the interaction tokens of all other passages. This new inter-passage attention pattern is somewhat similar to the attention pattern of the Fusion-in-Encoder model proposed for question answering [32]. The Fusion-in-Encoder model enables passage interactions within an encoder by adding additional so-called global tokens to which all passage tokens can attend to, and the global tokens can attend to all passage tokens. However, no direct interaction between passage tokens is possible. In contrast, our Set-Encoder allows for direct passage interactions as all passage tokens can attend to the interaction tokens of every other passage.

3 The Set-Encoder Model

Our Set-Encoder introduces inter-passage attention, a novel attention pattern to model permutation-invariant interactions between passages. Inter-passage attention addresses three challenges: (1) input 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 must be “lightweight” enough for efficient fine-tuning.

Previous work has addressed the first challenge (attention between passages) by concatenating the query and multiple passages into one input sequence, visualized in Figure 1 (bottom). However, concatenation violates the second challenge: the language model's positional encodings (used to determine a token's position) span across passage boundaries and thus encode the order of passages in the sequence. 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, our new inter-passage attention pattern processes the passages in parallel with individual input sequences per passage, as visualized in Figure 1 (top). Each input sequence's positional encodings start from zero, so no information about the passage order is encoded (second challenge). To still let the passages attend to each other (first challenge), we use a special passage interaction [INT] token that aggregates semantic information about its passage and to which tokens from other sequences can attend. The Set-Encoder allows passage interactions solely through these [INT] tokens, making inter-passage attention computationally efficient (third challenge).

3.1 Permutation-Invariant Input Encoding

Standard pointwise cross-encoders compute a relevance score for query–passage pairs (q, d) given as token sequences $t_{1_q} \dots t_{m_q}$ and $t_{1_d} \dots t_{m_d}$. In this work we use BERT-style encoding [23], meaning the final tokens t_{m_q} and t_{m_d} are special [SEP] separator tokens. The concatenated sequence is prepended by a special [CLS] classification token t_c. The resulting input sequence $t_c\, t_{1_q} \dots t_{m_q}\, t_{1_d} \dots t_{m_d}$ is then passed through a transformer-based encoder model. Finally, the relevance score of d for q is computed by a linear transformation on the final contextualized embedding of the [CLS] token t_c.

The Set-Encoder instead receives 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, an [INT] 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 [INT] 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 applying a linear transformation to the passage's final [CLS] token embedding.

Our batched input encoding is permutation-invariant because each input sequence's positional encodings start from zero. For example, the [INT] token is always at position 1 and 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

The Set-Encoder allows every sequence to attend to the [INT] tokens of every other sequence. Our intuition is that these [INT] tokens aggregate the semantic information from their sequence and can share it with all other sequences.

This hypothesis is motivated by the token-based semantic information aggregation in many model architectures. For instance, in bi-encoders, the [CLS] token captures a query or passage's semantic information, and in a standard pointwise cross-encoder, the [CLS] token aggregates query–passage information to compute a relevance score. We hypothesize that our additional [INT] token can also aggregate semantic information and share this information with other passages. We also tested directly using the [CLS] token for passage interactions and found this variant of passage interaction to produce effective re-rankers but being incapable of learning passage interactions (see Section 4.3). To implement inter-passage attention, we modify the original attention function of the transformer [65]:

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

where Q, K, and V are matrices of embedding vectors, in which each column vector corresponds to a token from the encoder's input sequence.

The inter-passage attention's input has 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 [INT] tokens of all other input sequences, we append the [INT] token's embedding vectors from the other input sequences' embedding matrices $K^{(j)}$ and $V^{(j)}$ to the input sequence's embedding matrices $K^{(i)}$ and $V^{(i)}$.

Specifically, let $\bar{K}^{(i)} = [K_2^{(j)} : j \neq i]$ and $\bar{V}^{(i)} = [V_2^{(j)} : j \neq i]$ be the matrices concatenating the [INT] token's embedding vectors from $K^{(j)}$ and $V^{(j)}$ of every passage $d_j \neq d_i$, 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'). Appending the [INT] tokens from the other passages' embedding vectors to the sequence matrices, we obtain the new inter-passage attention pattern as: $\text{Attention}(Q^{(i)}, [K^{(i)}\bar{K}^{(i)}], [V^{(i)}\bar{V}^{(i)}])$.

3.3 Fine-tuning

We follow Schlatt et al. [60] to fine-tune the Set-Encoder in two stages. In the first stage, the model is fine-tuned on MS MARCO [40] relevance labels using InfoNCE loss [44]:

$$\mathcal{L}_{\text{InfoNCE}} = -\log\frac{\exp(s_+)}{\sum_{i=1}^{k}\exp(s_i)}\,.$$

For a single query q, the loss is computed over the predicted relevance scores s_i for a set of passages $\{d_1, \ldots d_k\}$ of which only one is a labeled relevant passage d_+. The other $k-1$ passages are sampled from the top-200 passages retrieved by ColBERTv2 [58] for q.

In the second stage, the model is distilled from LLM-based cross-encoder rankings using the Rank-DistiLLM dataset [60] and the RankNet loss [4]:

$$\mathcal{L}_{\text{RankNet}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}_{r_i < r_j} \log(1 + e^{s_i - s_j}) \,,$$

where $\mathbb{1}$ is the indicator function and r_i is the rank the passage d_i gets assigned by a RankZephyr [53] teacher model, a state-of-the-art LLM-based cross-encoder.

3.4 Fine-tuning to Detect Duplicates

In experiments on TREC Deep Learning and TIREx we found the Set-Encoder does not need passage interactions to be effective and, therefore, does not make use of the [INT] token to share information between passages (details in Section 4.2). To ensure that the model actually learns passage interactions, we modify the first-stage fine-tuning to include an inter-passage duplicate detection loss. We randomly sample a passage $d_\times$ from the set of input passages, duplicate it, and add it to the set of input passages as passage d_{k+1}. We then add a binary classification head to the model that outputs a probability p_i of whether passage d_i is the duplicate passage. As a duplicate-aware InfoNCE, we use InfoNCE plus the binary cross-entropy loss over the model's duplication probabilities:

$$\mathcal{L}_{\text{DA-InfoNCE}} = \mathcal{L}_{\text{InfoNCE}} + \sum_{i=1}^{k} \mathbb{1}_{d_i = d_\times} \log p_i + \mathbb{1}_{d_i \neq d_\times} \log(1 - p_i) \,.$$

3.5 Fine-tuning to Rank According to Novelty

Finally, to demonstrate advantages of listwise models, we also fine-tune the Set-Encoder to re-rank based on passage relevance and novelty (i.e., whether a passage provides novel information compared to above ranked ones [8]). As there are no large-scale datasets for this task (e.g., the diversity portion of the TREC Web tracks is too small for fine-tuning), we create the new Rank-DistiLLM-Novelty dataset exploiting that MS MARCO, the basis for Rank-DistiLLM, contains many near-duplicate passages. In a ranking from the Rank-DistiLLM dataset, we group passages into a "novelty group" that scikit-learn's [48] agglomerative clustering combines with a word-based Jaccard similarity > 0.5. We treat the TREC 2019 and 2020 Deep Learning data analogously for evaluation [21, 22].

For novelty-aware re-ranking, we propose a new novelty-aware RankNet loss function. Intuitively, the loss penalizes a model if it ranks a less relevant passage higher than a more relevant one or a near-duplicate passage higher than a non-duplicate one. Let c_i be the cluster of near-duplicate passages to which d_i belongs. The novelty-aware RankNet loss then is

$$\mathcal{L}_{\text{NA-RankNet}} = \sum_{i=1}^{k} \sum_{j=1}^{k} \mathbb{1}_{\bar{r}_i < \bar{r}_j} \log(1 + e^{s_i - s_j}) \,,$$

with $\bar{r}_i = r_i \cdot (1 - \max_{j=1\ldots k} \mathbb{1}_{c_i = c_j} \cdot \mathbb{1}_{s_i < s_j})$ denoting adjusted relevance labels that are set to zero if the model already has ranked a near-duplicate higher.

4 Evaluation

To evaluate the effectiveness of the Set-Encoder, we conduct experiments on the TREC Deep Learning (DL) 2019 and 2020 passage tracks [21, 22] and on all tasks from the 13 collections in the TIREx platform [1–3, 5, 9–22, 25, 28–30, 55, 56, 67–71]. For TREC DL, we report nDCG@10 when re-ranking the top-100 passages retrieved by either BM25 [57] or ColBERTv2 [58]. For the novelty-based ranking, we also report α-nDCG@10 [8] and consider all passages from a near-duplicate cluster to belong to a subtopic (cf. Section 3.5 for the clustering). We set $\alpha = 0.99$, so that only the first passage from a subtopic contributes to the gain. For TIREx, we report nDCG@10 micro-averaged across all queries associated with a collection and we report the macro-averaged arithmetic and geometric mean across all collections.

We compare the Set-Encoder with RankGPT-4o [61] and RankZephyr [53], two state-of-the-art listwise LLM-based cross-encoders. As GPT-4o has a maximum input length of 128,000 tokens, we also test re-ranking 100 passages at once (RankGPT-4o Full). Additionally, we include LiT5-Distill [62], monoT5 3B [42], and RankT5 3B [73] cross-encoders, and a pointwise monoELECTRA model as it basically is the Set-Encoder without inter-passage attention [60].

4.1 Experimental Setup

We mostly follow Schlatt et al. [60] for fine-tuning the Set-Encoder and monoELECTRA models. We truncate queries to at most 32 tokens and passages to 256 tokens and use $\text{ELECTRA}_{\text{BASE}}$[1] and $\text{ELECTRA}_{\text{LARGE}}$[2] [7] checkpoints from HuggingFace [72] as starting points for fine-tuning. We use the relevance labels from the MS MARCO passage dataset [40] and the ColBERTv2 [58] hard negatives from Rank-DistiLLM [60] for first-stage fine-tuning. We fine-tune all models for 20k steps with a batch size of 32 using InfoNCE loss and 7 negatives per query. We then use the ColBERTv2–RankZephyr distillation scores from RankDistiLLM for second-stage fine-tuning with a batch size of 32 and 100 passages per query and fine-tune for 3 epochs using RankNet loss. For both fine-tuning stages we use the AdamW [36] optimizer with a learning rate of 10^{-5}.

We also fine-tune the Set-Encoder in the first-stage using our newly proposed duplicate-aware InfoNCE for at least 20k steps but at most 60k steps or until the duplicate-detection cross-entropy loss is smaller than 0.05 for at least 100 steps. For the novelty ranking task, we fine-tune models for 10 epochs using our novelty-aware RankNet loss, using early stopping and α-nDCG@10 on the TREC Deep Learning 2021 task as the criterion.

All models were fine-tuned on a single NVIDIA A100 40GB GPU and we used the following packages and frameworks: ir_datasets, ir-measures, Jupyter, Lightning, Lightning IR, matplotlib, NumPy, pandas, PyTerrier, PyTorch, and SciPy [24, 27, 31, 33, 37–39, 47, 59, 64, 66].

[1] `google/electra-base-discriminator`
[2] `google/electra-large-discriminator`

Table 1: Effectiveness (nDCG@10) of various cross-encoders on the TREC Deep Learning 2019 and 2020 tracks using BM25 or ColBERTv2 as the first stage retrieval. The highest and second-highest scores per setting are bold and underlined, respectively. Model sizes given as number of parameters. † denotes a significant difference ($p < 0.05$, Holm-Bonferroni-corrected) to the Set-Encoder with 330M parameters.

Model	Size	TREC DL 19		TREC DL 20	
		BM25	CBv2	BM25	CBv2
First Stage	–	0.480†	0.732†	0.494†	0.724†
RankGPT-4o	N/A	0.725	0.784	0.719	0.793
RankGPT-4o Full	N/A	<u>0.732</u>	0.781	0.711	0.796
RankZephyr	7B	0.719	0.749	0.720	<u>0.798</u>
LiT5-Distill	220M	0.696	0.753	0.679†	0.744†
monoT5 3B	3B	0.705	0.745	0.715	0.757
RankT5 3B	3B	0.710	0.752	0.711	0.772
monoELECTRA	110M	0.720	0.768	0.711†	0.770
	330M	**0.733**	0.765	<u>0.727</u>	**0.799**
Set-Encoder	110M	0.724	<u>0.788</u>	0.710†	0.777
	330M	0.727	**0.789**	**0.735**	0.790

4.2 Effectiveness on Cranfield-Style Datasets

In-domain Re-ranking. Table 1 shows the models' effectiveness on TREC DL. The Set-Encoder is on par with the state-of-the-art cross-encoders—the 330M parameter variant even is the most effective in two of four retrieval settings and the differences to the other cross-encoders is not significant in any of the scenarios (paired t-test with $p < 0.05$ and Holm-Bonferroni correction). Interestingly, the pointwise monoELECTRA model is similarly effective and the most effective model in the other two of the four settings. This suggests that passage interactions are unnecessary in the TREC DL scenarios; a stark contrast to popular belief and previous work stating that passage interactions are beneficial for re-ranking [42, 43, 51, 53, 62, 73].

Out-of-domain Re-ranking. The in-domain findings also hold in out-of-domain scenarios. Table 2 shows the effectiveness on the 13 TIREx collections (RankGPT excluded as TIREx does not allow external API calls) and the Set-Encoder is again on par with the state-of-the-art RankZephyr. In some cases RankZephyr is more effective in other cases the Set-Encoder. The differences are significant in some cases, but on average both models are similarly effective. The same holds true for the large monoELECTRA model.

Passage Interaction "Unneccessity". The on-par monoELECTRA effectiveness in the above experiments suggests that passage interactions are not really necessary. We think that there are three main reasons. First, we have fine-tuned monoELECTRA with the best available data [60], second, the TREC DL and TIREx scenarios come with relevance judgments that are independent of each other, and third, the standard evaluation schemes do not discount near-duplicate

Table 2: Effectiveness (nDCG@10) of various cross-encoders micro-averaged for test collections from the TIREx framework [25]. The macro-averaged geometric means (column 'G. Mean') are computed across all collections. Model sizes given as the number of parameters. The highest and second-highest averaged scores per collection are bold and underlined, respectively. † denotes a significant difference ($p < 0.05$, Holm-Bonferroni-corrected) to the Set-Encoder with 330M parameters.

Model	Size	Antique	Args.me	ClueWeb09	ClueWeb12	CORD-19	Cranfield	Disks4+5	GOV	GOV2	MEDLINE	NFCorpus	Vaswani	WaPo	G. Mean
First Stage	–	.510†	.405	.174†	**.344**†	.586†	**.012**	.424†	.332†	.467†	.385	.268†	.447†	.364†	.289
RankZephyr	7B	.528†	.364†	.220	.287	**.767**†	.009	.542	.423	.560	**.461**†	.299	.512	**.508**	.322
LiT5-Distill	220M	.571†	.395	.223	.259†	.686	.011	.495†	.385†	.534†	.354†	.278†	.429†	.470	.305
monoT5 3B	3B	.537†	.392	.202†	.268	.603†	.011	.545	.407	.514†	.378	**.308**	.458†	.476	.307
RankT5 3B	3B	.592	**.421**	.228	.326	.713	.010	.538	.415	.528†	.406	.307	.459†	.468†	.324
m.ELECTRA	110M	.587	.375†	.219†	.288	.692	.010	.507†	.388†	.541†	.399	.291	.522	.458†	.314
	330M	.570†	.369†	.230	.301	.716	.008	**.546**	**.424**	.572	.419	.301	.526	.504	.321
Set-Encoder	110M	.588	.375†	.220†	.285	.683	.010	.513†	.389†	.543†	.396	.291	.523	.461†	.314
	330M	**.600**	.409	**.242**	.297	.702	.009	.534	.418	**.573**	.405	.297	**.530**	**.508**	**.325**

results. As the assessors and evaluation treat passages independent of each other, re-rankers very likely can also score passages individually and still be effective. Passage interactions are beneficial in tasks where, for example, fairness, diversity, or novelty are considered, and we investigate this in more detail in Section 4.3.

Permutation Invariance. Additionally, despite being fine-tuned on RankZephyr rankings, the Set-Encoder and monoELECTRA are more effective in most out-of-domain re-ranking tasks. The Set-Encoder achieves a higher effectiveness than RankZephyr on 6 out of 13 corpora, RankZephyr is more effective on 5 corpora, and they reach same effectiveness on 2. Also monoELECTRA is more effective than RankZephyr on 9 corpora and less effective on only 4. RankZephyr only reaches a comparable mean effectiveness by being substantially more effective on the two medical corpora, CORD-19 and MEDLINE. We attribute the teacher model, RankZephyr, to be less effective than the student models to the fact that it is not permutation invariant. Consequently, RankZephyr is sensitive to the quality of the first-stage retrieval, which we investigate further in Section 4.4.

4.3 Novelty-Aware Ranking

To investigate the potential of passage interactions, we task the models to also take novelty into account (i.e., from a group of relevant but near-duplicate passages, only one should be ranked high). Using our novelty-aware RankNet loss (see Section 3.5), we fine-tune the Set-Encoder and monoELECTRA on novelty and we fine-tune the Set-Encoder on duplicate detection using our duplicate-aware InfoNCE loss (cf. Section 3.4; monoELECTRA is incapable of detecting duplicates). For the LLM-based cross-encoders, we augment the prompt to account for novelty. Table 3 shows the nDCG@10 and α-nDCG@10 of the models.

Table 3: Effectiveness (nDCG@10 and α-nDCG@10, $\alpha = 0.99$) of cross-encoders on the TREC Deep Learning 2019 and 2020 tracks clustered into near-duplicate subtopics. The highest and second-highest scores per setting are bold and underlined, respectively. † denotes a significant difference ($p < 0.05$, Holm-Bonferroni-corrected) to the Set-Encoder fine-tuned using $\mathcal{L}_{\text{DA-InfoNCE}}$ and then $\mathcal{L}_{\text{NA-RankNet}}$ (Row 14).

	Model	**Prompt / Loss**		**nDCG**		**α-nDCG**	
				2019	2020	2019	2020
(1)	First Stage	–		0.732	0.724	0.700†	0.722†
(2)	RankGPT-4o	Relevance		0.784†	0.793†	0.750	0.759
(3)		Novelty		0.778†	**0.806**†	0.741	<u>0.773</u>
(4)	RankGPT-4o Full	Relevance		0.781†	0.796†	0.738	0.763
(5)		Novelty		<u>0.785</u>†	<u>0.803</u>†	0.750	0.771
(6)	RankZephyr	Relevance		0.749	0.798†	0.699†	0.765
(7)		Novelty		0.753	0.800†	0.700†	0.760
(8)	**Model**	**1st $\mathcal{L}$**	**2nd $\mathcal{L}$**				
(9)	monoELECTRA	$\mathcal{L}_{\text{InfoNCE}}$	$\mathcal{L}_{\text{RankNet}}$	0.768†	0.770†	0.718†	0.745†
(10)			$\mathcal{L}_{\text{NA-RankNet}}$	0.704	0.675	<u>0.785</u>	0.753
(11)	Set-Encoder	$\mathcal{L}_{\text{InfoNCE}}$	$\mathcal{L}_{\text{RankNet}}$	0.780†	0.757†	0.733†	0.747†
(12)			$\mathcal{L}_{\text{NA-RankNet}}$	0.714	0.651	0.779	0.743†
(13)		$\mathcal{L}_{\text{DA-InfoNCE}}$	$\mathcal{L}_{\text{RankNet}}$	**0.788**†	0.777†	0.740†	0.752†
(14)			$\mathcal{L}_{\text{NA-RankNet}}$	0.710	0.690	**0.821**	**0.803**
(15)	Set-Enc. ~~[INT]~~	$\mathcal{L}_{\text{DA-InfoNCE}}$	$\mathcal{L}_{\text{NA-RankNet}}$	0.707	0.670	0.773	0.748†

The LLM-based cross-encoders are largely unaffected by the novelty prompt. Only RankGPT-4o Full has marginally but consistently higher α-nDCG@10 using the novelty prompt compared to the relevance prompt. One reason probably is the windowed ranking strategy of LLM-based cross-encoders. As the windowed strategy starts from the bottom of a ranking, initially highly ranked passages that are near-duplicates of other highly ranked passages simply cannot be moved to low ranks (cf. Section 4.4 for a more detailed analysis).

In contrast, the Set-Encoder improves considerably when fine-tuned using the novelty-aware NA-RankNet. It is significantly more effective in terms of α-nDCG@10 than its counterpart (compare rows 13 and 14) fine-tuned using normal RankNet. While not all α-nDCG@10 differences to the other cross-encoders are significant, the differences to the second-best models are sizable and are the largest between any two neighboring models when sorted by α-nDCG@10.

Looking at the InfoNCE-based fine-tuning, we find that the Set-Encoder fine-tuned with standard InfoNCE (row 12) is on par with a pointwise monoELECTRA model (row 10) in novelty-aware ranking. That is, the Set-Encoder is unable to profit from inter-passage attention when first fine-tuned to only rank according to relevance. However, when initially fine-tuned using duplicate-aware DA-InfoNCE (i.e., when the model learns to rank according to relevance and to detect exact duplicates; row 14), the Set-Encoder profits from inter-passage attention and is substantially more effective. We interpret this as further evidence

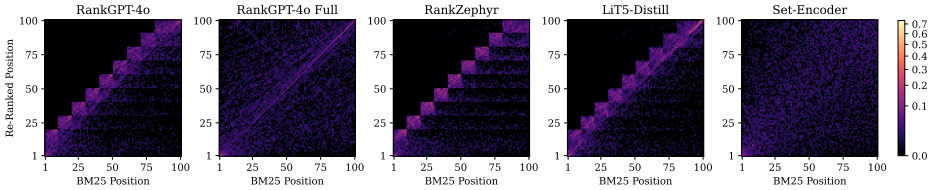

Fig. 2: Proportional rank changes of various cross-encoders for re-ranking BM25 averaged across all queries from the TREC Deep Learning 2019 and 2020 tracks.

that typical TREC-style relevance assessments and evaluation setups do not contain a training signal which necessitates passage interactions. Therefore, trained on such data, the Set-Encoder does not learn to use inter-passage attention. Only when the model is specifically "primed" to use inter-passage attention, by being fine-tuned to detect duplicates, can it learn passage interactions and then improve on novelty ranking.

[INT] Ablation. To determine the effect of the [INT] token, we compare the Set-Encoder's effectiveness when using the [INT] token and when using the [CLS] token for interaction. The results in the row 15 of Table 3 indicate that the model using the [CLS] token does not model passage interactions since it is on par with the pointwise monoELECTRA (row 10). In fact, using the [CLS] token instead of the [INT] token, the model is already unable to learn to detect duplicates during the initial fine-tuning. We hypothesize that the [CLS] token is not suited to model passage interactions as it has been heavily pre-trained and thus is not flexible enough to learn new interactions in fine-tuning. Further investigating this hypothesis could be an interesting direction for future work.

4.4 Permutation Invariance

Positional Biases. All previous listwise cross-encoders have an implicit positional bias by concatenating the input passages. Most LLM-based cross-encoders also have an explicit positional bias due to the limited input length. For example, to re-rank 100 passages, the original RankGPT model, RankZephyr, and LiT5-Distill use an overlapping windowed strategy. Starting from the bottom of the ranking, the models re-rank the passages from positions 100 to 81, then from 90 to 71, and so on until the top 20 passages are re-ranked. We visualize these explicit and implicit biases in Figure 2 as the average proportional rank changes when re-ranking the top-100 passages retrieved by BM25 for TREC DL 2019 and 2020. The lighter a pixel, the more frequently a model ranks a passage from a particular position to another particular position.

For RankGPT-4o, RankZephyr, and LiT5-Distill, the explicit bias from the windowed strategy is immediately visible as a distinct step pattern. RankGPT-4o Full, which does not use the windowed strategy and re-ranks all 100 passages at once, does not exhibit the step pattern but instead has a distinct diagonal

Fig. 3: Effectiveness (nDCG@10) on different permutations of BM25 rankings for TREC Deep Learning 2019 and 2020.

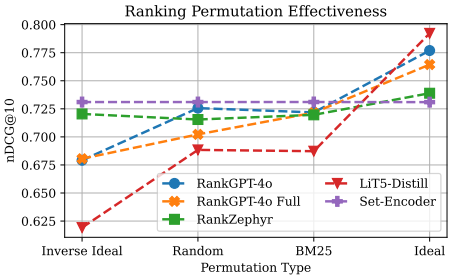

Table 4: Time (sec.) and memory footprint (GB) for re-ranking 100 passages.

Model	Size	Time	Memory
RankGPT-4o	N/A	18.773	N/A
RankGPT-4o Full	N/A	7.362	N/A
RankZephyr	7B	24.047	15.48
LiT5-Distill	220M	2.054	2.69
monoT5$_{3B}$	3B	0.998	29.36
RankT5$_{3B}$	3B	0.942	29.04
monoELECTRA	110M	0.139	1.18
	330M	0.215	2.69
Set-Encoder	110M	0.147	1.25
	330M	0.219	2.60

line that indicates that the model tends to keep passages in the same position as in the original ranking, caused by the model's auto-regressiveness. It is tasked to output the order of passages in the form of bracketed positions (e.g., `[2] > [1] > [3] > ...`) and has an implicit bias to simply output sequential positions.

By contrast, the Set-Encoder does not exhibit any immediately obvious positional biases; only a slight clustering of lighter pixels in the lower left corner and darker areas in the upper left and lower right show that the Set-Encoder's top-results correlate to some degree with the ones from the original BM25.

Ranking Perturbations. To test how the positional biases affect a cross-encoder's ranking effectiveness, we generate several "corrupted" perturbations of the top-100 passages retrieved by BM25 on the TREC DL 2019 and 2020 tracks: a random 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. Figure 3 visualizes the nDCG@10 of various LLM-based cross-encoders and our Set-Encoder on the different permutations.

The Set-Encoder is robust to the order of input passages and has the same effectiveness irrespective of the permutation. All other listwise cross-encoders are affected by the ranking permutations. RankGPT-4o, RankGPT-4o Full, and LiT5-Distill lose a substantial portion of their effectiveness when re-ranking the inverse ideal ranking and improve with increasingly better initial rankings. Only RankZephyr is comparatively robust to perturbations because it is fine-tuned to counteract positional biases, but it nonetheless fluctuates slightly.

Overall, the Set-Encoder reaches a higher nDCG@10 than the other cross-encoders on the inverse ideal, on the random, and on the original BM25 rankings. The LLM-based cross-encoders only achieve higher effectiveness when re-ranking the already ideal permutation. Due to the positional biases, most LLM-based cross-encoders require a good initial ranking to be effective. This insight explains how the Set-Encoder and monoELECTRA can manage to be more effective than their RankZephyr teacher model when re-ranking BM25 rankings for both the in-domain TREC Deep Learning scenarios and for most collections in the out-of-domain TIREx scenarios from Section 4.2.

5 Efficiency Analysis

As for the models' efficiency, we report inference time and GPU memory footprint when re-ranking TREC DL. The results in Table 4 show that the Set-Encoder adds only minimal overhead over monoELECTRA (i.e., over a similar model but without inter-passage attention). The Set-Encoder models are slightly slower than the respective monoELECTRA counterparts and the 110M parameter model uses marginally more memory. Interestingly, the 330M Set-Encoder uses slightly less memory than monoELECTRA, despite adding the additional [INT] interaction tokens. As this result is consistent across different batch sizes, we hypothesize that the CUDA-backend may be able to use more efficient kernels, but leave deeper investigations of the effect to future work.

Compared to state-of-the-art listwise LLM-based cross-encoders, the Set-Encoder substantially improves efficiency. While achieving comparable or better effectiveness (Sections 4.2 and 4.4), the 330M Set-Encoder is around 85 times faster than RankGPT-4o using a windowed strategy, about 33 times faster than RankGPT-4o re-ranking 100 passages at once, and about 110 times faster than RankZephyr. RankGPT-4o's memory consumption is not available, but compared to RankZephyr, the Set-Encoder uses 6 times less memory. Comparing with the smaller LLM-based LiT5-Distill cross-encoder, the Set-Encoder has the same memory footprint but is 9.5 times faster and substantially more effective.

6 Conclusion

In this paper, we have introduced the Set-Encoder: a new cross-encoder architecture that models passage interactions in a permutation-invariant way. Contrasting previous work, our empirical results show that passage interactions do not necessarily improve a cross-encoder's re-ranking effectiveness. A pointwise model fine-tuned using current distillation methods is as effective as state-of-the-art LLM-based cross-encoders in typical Cranfield-style settings in which the passages were judged independent of each other so that passage interactions are not important for effective re-ranking. But when additionally considering inter-document related aspects such as novelty, the Set-Encoder is more effective than its pointwise counterpart and previous state-of-the-art listwise cross-encoders.

Our results also show that permutation invariance is very helpful for an efficient and effective re-ranking. Explicit and implicit positional biases in previous listwise cross-encoders lead to inconsistent and suboptimal effectiveness across different settings. Using our new permutation-invariant inter-passage attention pattern, the Set-Encoder is on par or more effective than previous listwise cross-encoders in all ranking settings. At the same time, the Set-Encoder is way more efficient in terms of both memory consumption and query latency.

Acknowledgements This publication has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070014 (OpenWebSearch.EU, https://doi.org/10.3030/101070014).

References

[1] Bondarenko, A., Fröbe, M., Kiesel, J., Syed, S., Gurcke, T., Beloucif, M., Panchenko, A., Biemann, C., Stein, B., Wachsmuth, H., Potthast, M., Hagen, M.: Overview of Touché 2022: Argument Retrieval. In: Proceedings of CLEF 2022, pp. 311–336 (2022), https://doi.org/10.1007/978-3-031-13643-6_21

[2] Bondarenko, A., Gienapp, L., Fröbe, M., Beloucif, M., Ajjour, Y., Panchenko, A., Biemann, C., Stein, B., Wachsmuth, H., Potthast, M., Hagen, M.: Overview of Touché 2021: Argument Retrieval. In: Proceedings of CLEF 2021, pp. 450–467 (2021), https://doi.org/10.1007/978-3-030-85251-1_28

[3] Boteva, V., Gholipour, D., Sokolov, A., Riezler, S.: A Full-Text Learning to Rank Dataset for Medical Information Retrieval. In: Proceedings of ECIR 2016, pp. 716–722 (2016), https://doi.org/10.1007/978-3-319-30671-1_58

[4] Burges, C.J.C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to Rank Using Gradient Descent. In: Proceedings of ICML 2005, pp. 89–96 (2005), https://doi.org/10.1145/1102351.1102363

[5] Büttcher, S., Clarke, C.L.A., Soboroff, I.: The TREC 2006 Terabyte Track. In: Proceedings of TREC 2006 (2006), URL http://trec.nist.gov/pubs/trec15/papers/TERA06.OVERVIEW.pdf

[6] Buyl, M., Missault, P., Sondag, P.A.: RankFormer: Listwise Learning-to-Rank Using Listwide Labels. In: Proceedings of KDD 2023, pp. 3762–3773 (2023), https://doi.org/10.1145/3580305.3599892

[7] Clark, K., Luong, M.T., Le, Q.V., Manning, C.D.: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In: Proceedings of ICLR 2020 (2020), URL https://openreview.net/forum?id=r1xMH1BtvB

[8] Clarke, C.L., Kolla, M., Cormack, G.V., Vechtomova, O., Ashkan, A., Büttcher, S., MacKinnon, I.: Novelty and Diversity in Information Retrieval Evaluation. In: Proceedings of SIGIR 2008, pp. 659–666 (2008), https://doi.org/10.1145/1390334.1390446

[9] Clarke, C.L.A., Craswell, N., Soboroff, I.: Overview of the TREC 2004 Terabyte Track. In: Proceedings of TREC 2004 (2004), URL http://trec.nist.gov/pubs/trec13/papers/TERA.OVERVIEW.pdf

[10] Clarke, C.L.A., Craswell, N., Soboroff, I.: Overview of the TREC 2009 Web Track. In: Proceedings of TREC 2009 (2009), URL http://trec.nist.gov/pubs/trec18/papers/WEB09.OVERVIEW.pdf

[11] Clarke, C.L.A., Craswell, N., Soboroff, I., Cormack, G.V.: Overview of the TREC 2010 Web Track. In: Proceedings of TREC 2010 (2010), URL https://trec.nist.gov/pubs/trec19/papers/WEB.OVERVIEW.pdf

[12] Clarke, C.L.A., Craswell, N., Soboroff, I., Voorhees, E.M.: Overview of the TREC 2011 Web Track. In: Proceedings of TREC 2011 (2011), URL http://trec.nist.gov/pubs/trec20/papers/WEB.OVERVIEW.pdf

[13] Clarke, C.L.A., Craswell, N., Voorhees, E.M.: Overview of the TREC 2012 Web Track. In: Proceedings of TREC 2012 (2012), URL http://trec.nist.gov/pubs/trec21/papers/WEB12.overview.pdf

[14] Clarke, C.L.A., Scholer, F., Soboroff, I.: The TREC 2005 Terabyte Track. In: Proceedings of TREC 2005 (2005), URL http://trec.nist.gov/pubs/trec14/papers/TERABYTE.OVERVIEW.pdf

[15] Cleverdon, C.W.: The Significance of the Cranfield Tests on Index Languages. In: Proceedings of SIGIR 1991, pp. 3–12 (1991), https://doi.org/10.1145/122860.122861

[16] Collins-Thompson, K., Bennett, P.N., Diaz, F., Clarke, C., Voorhees, E.M.: TREC 2013 Web Track Overview. In: Proceedings of TREC 2013 (2013), URL http://trec.nist.gov/pubs/trec22/papers/WEB.OVERVIEW.pdf
[17] Collins-Thompson, K., Macdonald, C., Bennett, P.N., Diaz, F., Voorhees, E.M.: TREC 2014 Web Track Overview. In: Proceedings of TREC 2014 (2014), URL http://trec.nist.gov/pubs/trec23/papers/overview-web.pdf
[18] Craswell, N., Hawking, D.: Overview of the TREC-2002 Web Track. In: Proceedings of TREC 2002 (2002), URL http://trec.nist.gov/pubs/trec11/papers/WEB.OVER.pdf
[19] Craswell, N., Hawking, D.: Overview of the TREC 2004 Web Track. In: Proceedings of TREC 2004 (2004), URL http://trec.nist.gov/pubs/trec13/papers/WEB.OVERVIEW.pdf
[20] Craswell, N., Hawking, D., Wilkinson, R., Wu, M.: Overview of the TREC 2003 Web Track. In: Proceedings of TREC 2003, pp. 78–92 (2003), URL http://trec.nist.gov/pubs/trec12/papers/WEB.OVERVIEW.pdf
[21] Craswell, N., Mitra, B., Yilmaz, E., Campos, D.: Overview of the TREC 2020 Deep Learning Track. In: Proceedings of TREC 2020 (2020), URL https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf
[22] Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M.: Overview of the TREC 2019 Deep Learning Track. In: Proceedings of TREC 2019 (2019), URL https://trec.nist.gov/pubs/trec28/papers/OVERVIEW.DL.pdf
[23] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of NAACL 2019, pp. 4171–4186 (2019), https://doi.org/10.18653/v1/N19-1423
[24] Falcon, W., The PyTorch Lightning team: PyTorch Lightning (2023), https://doi.org/10.5281/zenodo.7859091
[25] Fröbe, M., Reimer, J.H., MacAvaney, S., Deckers, N., Reich, S., Bevendorff, J., Stein, B., Hagen, M., Potthast, M.: The Information Retrieval Experiment Platform. In: Proceedings of SIGIR 2023, pp. 2826–2836 (2023), https://doi.org/10.1145/3539618.3591888
[26] Gienapp, L., Fröbe, M., Hagen, M., Potthast, M.: Sparse Pairwise Re-ranking with Pre-trained Transformers. In: Proceedings of SIGIR 2022, pp. 72–80 (2022), https://doi.org/10.1145/3539813.3545140
[27] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array Programming with NumPy. Nature **585**(7825), 357–362 (2020), https://doi.org/10.1038/s41586-020-2649-2
[28] Hashemi, H., Aliannejadi, M., Zamani, H., Croft, W.B.: ANTIQUE: A Non-factoid Question Answering Benchmark. In: Proceedings of ECIR 2020, pp. 166–173 (2020), https://doi.org/10.1007/978-3-030-45442-5_21
[29] Hersh, W.R., Bhupatiraju, R.T., Ross, L., Cohen, A.M., Kraemer, D., Johnson, P.: TREC 2004 Genomics Track Overview. In: Proceedings of TREC 2004 (2004), URL http://trec.nist.gov/pubs/trec13/papers/GEO.OVERVIEW.pdf
[30] Hersh, W.R., Cohen, A.M., Yang, J., Bhupatiraju, R.T., Roberts, P.M., Hearst, M.A.: TREC 2005 Genomics Track Overview. In: Proceedings of TREC 2005 (2005), URL http://trec.nist.gov/pubs/trec14/papers/GEO.OVERVIEW.pdf
[31] Hunter, J.D.: Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering **9**(03), 90–95 (2007), ISSN 1521-9615, https://doi.org/10.1109/MCSE.2007.55

[32] Kedia, A., Zaidi, M.A., Lee, H.: FiE: Building a Global Probability Space by Leveraging Early Fusion in Encoder for Open-Domain Question Answering. In: Proceedings of EMNLP 2022, pp. 4246–4260 (2022), https://doi.org/10.18653/v1/2022.emnlp-main.285
[33] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., Team, J.D.: Jupyter Notebooks – A Publishing Format for Reproducible Computational Workflows. In: Positioning and Power in Academic Publishing: Players, Agents and Agendas, pp. 87–90, IOS Press (2016), https://doi.org/10.3233/978-1-61499-649-1-87
[34] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In: Proceedings of ICML 2019, pp. 3744–3753 (2019), URL https://proceedings.mlr.press/v97/lee19d.html
[35] Lin, J., Nogueira, R., Yates, A.: Pretrained Transformers for Text Ranking: BERT and Beyond. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers (2022), https://doi.org/10.1007/978-3-031-02181-7
[36] Loshchilov, I., Hutter, F.: Decoupled Weight Decay Regularization. In: Proceedings of ICLR 2019 (2019), URL https://openreview.net/forum?id=Bkg6RiCqY7
[37] MacAvaney, S., Macdonald, C., Ounis, I.: Streamlining Evaluation with `ir-measures`. In: Proceedings of ECIR 2022, pp. 305–310 (2022), https://doi.org/10.1007/978-3-030-99739-7_38
[38] MacAvaney, S., Yates, A., Feldman, S., Downey, D., Cohan, A., Goharian, N.: Simplified Data Wrangling with `ir_datasets`. In: Proceedings of SIGIR 2021, pp. 2429–2436 (2021), https://doi.org/10.1145/3404835.3463254
[39] Macdonald, C., Tonellotto, N., MacAvaney, S., Ounis, I.: PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In: Proceedings of CIKM 2021, pp. 4526–4533 (Oct 2021), https://doi.org/10.1145/3459637.3482013
[40] Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., Deng, L.: MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In: Proceedings of COCO@NeurIPS 2016 (2016), URL https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf
[41] Nogueira, R., Cho, K.: Passage Re-ranking with BERT. arXiv:1901.04085[v5] (2020), https://doi.org/10.48550/arXiv.1901.04085
[42] Nogueira, R., Jiang, Z., Pradeep, R., Lin, J.: Document Ranking with a Pretrained Sequence-to-Sequence Model. In: Findings of EMNLP 2020, pp. 708–718 (2020), https://doi.org/10.18653/v1/2020.findings-emnlp.63
[43] Nogueira, R., Yang, W., Cho, K., Lin, J.: Multi-Stage Document Ranking with BERT. arXiv:1910.14424 (2019), https://doi.org/10.48550/arXiv.1910.14424
[44] van den Oord, A., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv:1807.03748 (2019), https://doi.org/10.48550/arXiv.1807.03748
[45] Pang, L., Xu, J., Ai, Q., Lan, Y., Cheng, X., Wen, J.: SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval. In: Proceedings of SIGIR 2020, pp. 499–508 (2020), https://doi.org/10.1145/3397271.3401104
[46] Pasumarthi, R.K., Zhuang, H., Wang, X., Bendersky, M., Najork, M.: Permutation Equivariant Document Interaction Network for Neural Learning to

Rank. In: Proceedings of ICTIR 2020, pp. 145–148 (2020), https://doi.org/10.1145/3409256.3409819

[47] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Proceedings of NeurIPS 2019, pp. 8024–8035 (2019), URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[48] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011), https://doi.org/10.5555/1953048.2078195

[49] Pobrotyn, P., Bartczak, T., Synowiec, M., Białobrzeski, R., Bojar, J.: Context-Aware Learning to Rank with Self-Attention. In: Proceedings of eCom@SIGIR 2020 (2020), URL https://sigir-ecom.github.io/ecom2020/ecom20Papers/paper18.pdf

[50] Pradeep, R., Liu, Y., Zhang, X., Li, Y., Yates, A., Lin, J.: Squeezing Water from a Stone: A Bag of Tricks for Further Improving Cross-Encoder Effectiveness for Reranking. In: Proceedings of ECIR 2022, pp. 655–670 (2022), https://doi.org/10.1007/978-3-030-99736-6_44

[51] Pradeep, R., Nogueira, R., Lin, J.: The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. arXiv:2101.05667 (2021), URL http://arxiv.org/abs/2101.05667

[52] Pradeep, R., Sharifymoghaddam, S., Lin, J.: RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. arXiv:2309.15088 (2023), URL https://doi.org/10.48550/arXiv.2309.15088

[53] Pradeep, R., Sharifymoghaddam, S., Lin, J.: RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! arXiv:2312.02724 (2023), URL https://doi.org/10.48550/arXiv.2312.02724

[54] Reimers, N., Gurevych, I.: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of EMNLP-IJCNLP 2019, pp. 3980–3990 (2019), https://doi.org/10.18653/v1/D19-1410

[55] Roberts, K., Demner-Fushman, D., Voorhees, E.M., Hersh, W.R., Bedrick, S., Lazar, A.J.: Overview of the TREC 2018 Precision Medicine Track. In: Proceedings of TREC 2018 (2018), URL https://trec.nist.gov/pubs/trec27/papers/Overview-PM.pdf

[56] Roberts, K., Demner-Fushman, D., Voorhees, E.M., Hersh, W.R., Bedrick, S., Lazar, A.J., Pant, S.: Overview of the TREC 2017 Precision Medicine Track. In: Proceedings of TREC 2017 (2017), URL https://trec.nist.gov/pubs/trec26/papers/Overview-PM.pdf

[57] Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at TREC-3. In: Proceedings of TREC 1994, pp. 109–126 (1994), URL http://trec.nist.gov/pubs/trec3/papers/city.ps.gz

[58] Santhanam, K., Khattab, O., Saad-Falcon, J., Potts, C., Zaharia, M.: ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. arXiv:2112.01488 (2022), https://doi.org/10.48550/arXiv.2112.01488

[59] Schlatt, F., Fröbe, M., Hagen, M.: Lightning IR: Straightforward Fine-tuning and Inference of Transformer-based Language Models for Information Retrieval.

In: Proceedings of WSDM 2025 (2025), https://doi.org/10.1145/3701551.3704118, (to appear)
[60] Schlatt, F., Fröbe, M., Scells, H., Zhuang, S., Koopman, B., Zuccon, G., Stein, B., Potthast, M., Hagen, M.: Rank-DistiLLM: Closing the Effectiveness Gap Between Cross-Encoders and LLMs for Passage Re-Ranking. In: Proceedings of ECIR 2025 (2025), (to appear)
[61] Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D., Ren, Z.: Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In: Proceedings of EMNLP 2023, pp. 14918–14937 (2023), https://doi.org/10.18653/v1/2023.emnlp-main.923
[62] Tamber, M.S., Pradeep, R., Lin, J.: Scaling Down, LiTting Up: Efficient Zero-Shot Listwise Reranking with Seq2seq Encoder-Decoder Models. arXiv:2312.16098 (2023), https://doi.org/10.48550/arXiv.2312.16098
[63] Tang, R., Zhang, X., Ma, X., Lin, J., Ture, F.: Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models. arXiv:2310.07712 (2023), https://doi.org/10.48550/arXiv.2310.07712
[64] pandas development team, T.: Pandas-dev/pandas: Pandas. Zenodo (Apr 2024), https://doi.org/10.5281/zenodo.10957263
[65] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is All you Need. In: Proceedings of NeurIPS 2017, pp. 5998–6008 (2017), URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
[66] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**(3), 261–272 (2020), https://doi.org/10.1038/s41592-019-0686-2
[67] Voorhees, E., Alam, T., Bedrick, S., Demner-Fushman, D., Hersh, W.R., Lo, K., Roberts, K., Soboroff, I., Wang, L.L.: TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection. arXiv:2005.04474 (2020), https://doi.org/10.48550/arXiv.2005.04474
[68] Voorhees, E.M.: Overview of the TREC 2004 Robust Track. In: Procedings of TREC 2004 (2004), URL http://trec.nist.gov/pubs/trec13/papers/ROBUST.OVERVIEW.pdf
[69] Voorhees, E.M., Harman, D.: Overview of the Seventh Text REtrieval Conference (TREC-7). In: Proceedings of TREC 1998 (1998), URL http://trec.nist.gov/pubs/trec8/papers/overview_8.ps
[70] Voorhees, E.M., Harman, D.: Overview of the Eigth Text REtrieval Conference (TREC-8). In: Proceedings of TREC 1999 (1999), URL http://trec.nist.gov/pubs/trec8/papers/overview_8.ps
[71] Wang, L.L., Lo, K., Chandrasekhar, Y., Reas, R., Yang, J., Burdick, D., Eide, D., Funk, K., Katsis, Y., Kinney, R., Li, Y., Liu, Z., Merrill, W., Mooney, P., Murdick, D., Rishi, D., Sheehan, J., Shen, Z., Stilson, B., Wade, A., Wang, K., Wang, N.X.R., Wilhelm, C., Xie, B., Raymond, D., Weld, D.S., Etzioni, O., Kohlmeier, S.: CORD-19: The COVID-19 Open Research Dataset. arXiv:2004.10706 (2020), https://doi.org/10.48550/arXiv.2004.10706

[72] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 (2020), https://doi.org/10.48550/arXiv.1910.03771

[73] Zhuang, H., Qin, Z., Jagerman, R., Hui, K., Ma, J., Lu, J., Ni, J., Wang, X., Bendersky, M.: RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses. arXiv:2210.10634 (2022), https://doi.org/10.48550/arXiv.2210.10634