ABSTRACT
Pretrained language models such as BERT have been shown to be exceptionally effective for text ranking. However, there are limited studies on how to leverage more powerful sequence-to-sequence models such as T5. Existing attempts usually formulate text ranking as a classification problem and rely on postprocessing to obtain a ranked list. In this paper, we propose RankT5 and study two T5-based ranking model structures, an encoder-decoder and an encoder-only one, so that they not only can directly output ranking scores for each query-document pair, but also can be fine-tuned with “pairwise” or “listwise” ranking losses to optimize ranking performance. Our experiments show that the proposed models with ranking losses can achieve substantial ranking performance gains on different public text ranking data sets. Moreover, ranking models fine-tuned with listwise ranking losses have better zero-shot ranking performance on out-of-domain data than models fine-tuned with classification losses.

CCS CONCEPTS
• Information systems → Learning to rank.

KEYWORDS
T5, text ranking, ranking losses

1 INTRODUCTION
Text ranking is a fundamental component of countless real world applications such as search and question answering. Progress on pretrained language models in the past few years [7] and the release of large-scale public data sets [1, 20] enable a series of work [12, 23, 33] on text ranking models which directly encode textual query and document using pretrained language models, noticeably BERT [7]. Recently, large language models such as T5 [37], GPT-3 [2] and InstructGPT [34] have shown superior performance in various NLP tasks including sentiment analysis, coreference resolution, and translation. Such models often have much larger size available than previous models such as BERT [7] to store more hidden knowledge. They also mostly have a sequence-to-sequence interface to unify different NLP tasks from classification to text generation.

While BERT-based models have been well explored for text ranking [12, 23, 33], how to leverage T5 for text ranking is still under-explored and challenging. First, while many classification and text generation tasks fit into the sequence-to-sequence framework, text ranking tasks are more difficult: a text ranking model is often expected to output a numerical ranking score \( g \in \mathbb{R} \) for each query-document pair. Second, it is important to train a text ranking model with ranking losses [14, 25, 36] to optimize its ranking performance, where the losses take into account the ranking scores from multiple documents for each query. This is different from the typical T5 fine-tuning strategy where the objective is often formulated into a text generation loss for each single input sequence independently.

A typical approach to use T5 for text ranking is to convert the problem into a token generation problem. For example, Nogueira et al. [32] fine-tune the T5 model to predict a “true” or “false” token for a relevant or irrelevant query-document pair and then use a postprocessing step during inference to derive ranking scores to rank candidate documents. Such an approach can be considered as a “pointwise” classification formulation. How to extend this approach to fine-tune T5 with ranking losses is not well-explored.

In this paper, we propose RankT5 with the goal to support text ranking more natively with T5 by outputting ranking scores, instead of text tokens. We first adapt the encoder-decoder structure for this goal. We also propose an encoder-only structure which omits the T5 decoder. These two structure variants allow us to fine-tune T5 with various ranking losses to directly optimize ranking performance.

Experiments on MS MARCO and Natural Question (NQ) data sets show that our RankT5 models fine-tuned with specialized ranking losses can significantly outperform other T5 ranking models fine-tuned with classification losses and previously proposed T5 adaptations for ranking [32]. We also discover that models fine-tuned with some ranking losses tend to have better zero-shot performance than models fine-tuned with classification losses. Checkpoints of RankT5 fine-tuned with ranking losses are released publicly (Section 5).

2 RELATED WORK
Model structure. A typical ranking model structure design is the cross-attention model structure, where a query and a candidate document is concatenated into a sequence and fed into the model. This model structure has been explored for BERT-like encoder-only
model [10, 12, 33] and T5-like model [17, 32], but the model is not directly fine-tuned with ranking losses for the optimal ranking performance. There are also models taking multiple documents as input [29, 35, 48], but they are usually applied in a late ranking stage and are complementary to our work. Some other models rank documents by the likelihood of query generated from language models given the document [8, 17, 40, 50].

Notice that our focus is not on the retrieval task [9, 13, 18, 19, 24, 28, 41], where the model needs to score hundreds of millions of documents in the entire corpus almost instantly for each query. **Fine-tuning with ranking losses.** Early explorations [32, 33] of applying pretrained language models on the document reranking task mainly use "pointwise" losses, where the loss is calculated for each query-document pair independently. There are some recent works that use a "listwise" loss [10, 12, 27, 38], but they only fine-tune BERT, RoBERTa, ERNIE, etc. There is no existing work fine-tuning sequence-to-sequence models like T5 with ranking losses. Others fine-tune retrieval models with pairwise or softmax loss [18, 26, 47], while we focus on reranking models in this work.

There are also several studies of pretraining methods tailored to enhance text ranking [11, 30, 44–46]. While we only focus on fine-tuning, our proposed method can be seamlessly applied.

## 3 PRELIMINARIES

**Problem definition.** We provide the formalized definition of a ranking task. For each query \( q_i \), a list of \( m \) candidate documents \( D_i = \{d_{i1}, \ldots, d_{im}\} \) are provided, which are usually the output from a retriever. The relevance labels of candidate documents with regard to the query are represented as \( y_i = (y_{i1}, \ldots, y_{im}) \) where \( y_{ij} \) \( \geq 0 \).

The objective is to train a ranking model \( f \) which takes a query-document pair as input and outputs a ranking score \( \hat{y}_{ij} = f(q_i, d_{ij}) \in \mathbb{R} \). We aim to optimize the ranking metrics after we sort the documents in \( D_i \) for each query \( q_i \) based on their ranking scores.

**T5.** T5 [37] is a text-to-text pretrained generative language model with an encoder-decoder structure. It takes a piece of text as input and outputs a sequence of text tokens in an autoregressive manner.

More formally, we denote the input to the T5 encoder as a text sequence \( s = \{w_1, \ldots, w_l\} \) and the previously generated tokens from the T5 decoder as \( t_{1:k-1} \) during the autoregressive decoding process. We formalize the T5 model structure as:

\[
P_k = T5(s, t_{1:k-1}) = \text{Softmax}(\text{Dense}(\text{Dec}(\text{Enc}(s), t_{1:k-1})))
\]

where the output is a vector with the length of the vocabulary size \( p_k \in \mathbb{R}^{|V|} \), representing the predictive probability of each token in the vocabulary being generated at the \( k \)-th position. \( \text{Enc}(\cdot) \) and \( \text{Dec}(\cdot) \) are the encoder and the decoder of T5 respectively; \( \text{Dense}(\cdot) \) is a dense layer; \( \text{Softmax}(\cdot) \) is a softmax transformation layer that normalizes the vector into a probability distribution.

## 4 RANKT5 MODELING

### 4.1 Model structure

We propose to directly obtain the numerical ranking score as the model output, so that the model can be directly fine-tuned by ranking losses to optimize ranking metrics. We present two variants based on T5: an encoder-decoder and an encoder-only model.

Figure 1 summarizes the proposed model structures. **Input sequence.** For each candidate document \( d_{ij} \) and its query \( q_i \), we concatenate them with prefix "Document:" and "Query:" respectively to construct the input text \( s_{ij} \):

\[
s_{ij} = \text{Query: } q_i \text{ Document: } d_{ij}
\]

The construction of input sequence is similar to Nogueira et al. [32] except that we do not include the "Relevant:" postfix. The postfix does not affect the results in our experiments.

**Encoder-decoder (EncDec).** This model variant is a simple adaptation of the T5 model by using the first output token of the decoder. In this variant, we feed the input into a T5 model and obtain the unnormalized logits \( z \) over the entire vocabulary:

\[
z = \text{Dense}(\text{Dec}(\text{Enc}(s_{ij})))
\]

Notice that we omit the softmax layer over vocabulary so that the elements in \( z \) can be arbitrary real numbers. We also do not need previous tokens \( t_{1:k-1} \) since we do not generate tokens.

We specify an unused token in T5 vocabulary "\text{extra\_id\_10}" and take its corresponding unnormalized logits as the ranking score:

\[
\hat{y}_{ij} = z_{\langle \text{extra\_id\_10} \rangle}
\]

where we use the notation \( z_{\langle \cdot \rangle} \) to represent the logits corresponding to the token \( w \in V \). The special token can be any other unused token in the vocabulary. An illustration of this model structure can be found in Figure 1(a).

**Encoder-only (Enc).** We also propose an encoder-only variant since we do not need to perform autoregressive decoding. The input text \( s_{ij} \) remains the same as the encoder-decoder model. We take the output of the encoder \( \text{Enc}(s_{ij}) \), which is a sequence of embedding vectors \( \{e_1, \ldots, e_l\} \), and apply a pooling layer \( \text{Pool}(\cdot) \) to aggregate them into a single embedding vector. Then we apply the dense layer which directly projects the embedding vector to the ranking score \( \hat{y}_{ij} \). More formally,

\[
\hat{y}_{ij} = \text{Dense}(\text{Pool}((\text{Enc}(s_{ij}))))
\]

Figure 1(b) summarizes the proposed model structure.

### 4.2 Training

For each query \( q_i \) and a list of its candidate documents \( D_i \), we obtain the list of predicted ranking scores \( \hat{y}_i \) by applying the model on each query-document pair \( (q_i, d_{ij}) \) where \( d_{ij} \in D_i \). Then we can train the model by optimizing a ranking-based training loss.
function \(\ell(y_i, \hat{y}_i)\), defined on the two lists of predicted scores \(\hat{y}_i\) and relevance labels \(y_i\). We study the following ranking losses:

**Pairwise logistic (Pair).** We can train the model using a pairwise logistic ranking loss [4]:

\[
\ell_{\text{Pair}}(y_i, \hat{y}_i) = \sum_{j=1}^{m} \sum_{j' > j} I_{y_i} > y_{i'} \cdot \log(1 + e^{\hat{y}_i - \hat{y}_{i'}})
\]

where the ranking problem is converted into a binary classification problem on the order of each candidate document pair in the same list with different relevance labels.

**Listwise softmax cross entropy (Softmax).** We can also define a listwise softmax cross entropy loss [3], which is a simple version of ListNet [5]. It takes the entire list into account.

\[
\ell_{\text{Softmax}}(y_i, \hat{y}_i) = - \sum_{j=1}^{m} y_{ij} \log \left( \frac{e^{\hat{y}_{ij}}}{\sum_{j'} e^{\hat{y}_{ij'}}} \right)
\]

**Listwise poly-1 softmax cross entropy (Poly1).** We also try a recently proposed extended version of the softmax cross-entropy loss called PolyLoss [22]. The idea is to adjust the weights of polynomial terms in the Taylor expansion of a softmax cross-entropy loss. A simplified version only adjusted the first polynomial term:

\[
\ell_{\text{Poly1}}(y_i, \hat{y}_i) = \ell_{\text{Softmax}}(y_i, \hat{y}_i) + \sum_{j=1}^{m} \epsilon \cdot y_{ij} \left( 1 - \frac{e^{\hat{y}_{ij}}}{\sum_{j'} e^{\hat{y}_{ij'}}} \right)
\]

where \(\epsilon\) is a parameter specified by users, representing how much extra coefficient to be placed for the first polynomial term.

5 EXPERIMENT SETUP

5.1 Data sets

**MS MARCO.** We use the MS MARCO passage ranking data set [1]. The data set contains around 530,000 queries in the “train” partition and around 6,800 queries in the “dev” partition. The candidates are from a corpus with more than 8.8 million passages. For each query, relevant passages are labeled as 1, and others are labeled as 0. We use a dual-encoder retriever [31] fine-tuned on MS MARCO to retrieve the top-1000 passages for each query in both the “train” and “dev” partitions as the candidate documents.

Notice that in this paper, the term “document” is used interchangeably with “passage”. We do not focus on long document ranking tasks such as the MS MARCO document ranking task.

**Natural Questions (NQ).** We use the Natural Questions data set [20] with more than 50,000 queries in the “train” partition and 8,000 in the “dev” partition. We adopt the preprocessing setup similar to Karpukhin et al. [18] to construct the corpus of passages. Similar to MS MARCO, binary relevance labels are provided. We use a dual-encoder retriever [26] fine-tuned on NQ to retrieve the top-1000 passages for each query.

**Training data construction.** We construct the training data by first selecting a document with label 1 for each query and then uniformly randomly sampling \((m - 1)\) documents from the top-1000 retrieved documents with label 0. We set the list size \(m\) to 36 in our experiments due to hardware constraints.

For models with pointwise training losses, we upsample documents with label 1 in each query to the same number as documents with label 0 in order to achieve the optimal performance.

**Evaluation.** We evaluate the performance on the “dev” partition on both data sets. We perform model inference on the top-1000 documents retrieved by the dual-encoder retriever of each data set respectively. We evaluate the performance by Mean Reciprocal Rank (MRR@10) [43], Normalized Discounted Cumulative Gain (NDCG@5, 10) [16] and Mean Average Precision (MAP).

5.2 Parameter configurations

We initialize the ranking model with pretrained T5-Large checkpoint if not specified otherwise. For the pooling layer in RankT5-Enc, we follow BERT [7] and take the embedding vector of the first token. The results do not differ when using other pooling methods like mean pooling in our experiments.

We set the maximum input sequence length to 128 and 128+256 = 384 for MS MARCO and NQ respectively. We do not find significant performance degradation compared to using 512 as the sequence length. The batch size is set to 32 lists per batch for both data sets. We use a constant learning rate of \(1 \times 10^{-4}\) during fine-tuning. For the MS MARCO data set, we fine-tune our models for 50,000 steps. For the NQ data set, we fine-tune most of our models for 100,000 steps, except the ones using pointwise cross-entropy loss, which achieves the best performance at 25,000 steps. For Poly1 loss, we simply set \(\epsilon = 1\) for all of our experiments.

The model implementation is based on T5X\(^1\). All the ranking losses are implemented in Rax\(^2\) [15]. Selected checkpoints of fine-tuned RankT5 are released publicly\(^3\).

6 RESULTS

**Overall comparison.** We compare the performance of our proposed rankers with different model structures and different training losses. We train the monoT5 model [32] and BERT [12] models on our data sets and report their performance as baselines. The BERT models are initialized from BERT-Large checkpoints and fine-tuned with pointwise cross-entropy classification loss (PointCE) and the listwise softmax cross-entropy ranking loss (Softmax) respectively. We also include results where our proposed RankT5 models are fine-tuned with the pointwise cross-entropy classification loss. All results are presented in Table 1.

We observe that fine-tuning with ranking losses substantially helps RankT5 improve the ranking performance. RankT5 with Softmax and Poly1 consistently outperform other baselines, including RankT5 with the classification loss (PointCE). On both data sets, the best performing RankT5 improves around +2% in all metrics compared to monoT5.

We also verify that the initial checkpoint of T5 shows advantage over the initial checkpoint of BERT. RankT5-Enc and BERT have similar model size and structure but different pretrained checkpoints. When fine-tuned with the Softmax loss, RankT5-Enc outperforms BERT with a large margin on both data sets (+3.7% to +4.7% in MRR@10). This demonstrates the importance of using state-of-the-art initial checkpoints and justifies the necessity of this work to adapt sequence-to-sequence models like T5 for ranking.

---

\(^{1}\)https://github.com/google-research/t5x

\(^{2}\)https://github.com/google/rax

\(^{3}\)https://github.com/google-research/google-research/tree/master/rankt5
Table 1: Comparing ranking performances of different ranking models. The best performance for each data set is bolded. Results with † are statistically significantly ($p \leq 0.05$) better than monoT5.

<table>
<thead>
<tr>
<th>Model</th>
<th>Loss</th>
<th>MS MARCO</th>
<th>NQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MRR@10</td>
<td>NDCG@5</td>
</tr>
<tr>
<td>BERT</td>
<td>PointCE</td>
<td>0.3867</td>
<td>0.4127</td>
</tr>
<tr>
<td></td>
<td>Softmax</td>
<td>0.3928</td>
<td>0.4173</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5157</td>
<td>0.5515</td>
</tr>
<tr>
<td>monoT5</td>
<td>Generation</td>
<td>0.4156</td>
<td>0.4448</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5406</td>
<td>0.5861</td>
</tr>
<tr>
<td>RankT5-EncDec</td>
<td>PointCE</td>
<td>0.4069</td>
<td>0.4096</td>
</tr>
<tr>
<td></td>
<td>Softmax</td>
<td>0.3997</td>
<td>0.4014†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5403</td>
<td>0.5383</td>
</tr>
<tr>
<td>RankT5-Enc</td>
<td>PointCE</td>
<td>0.4216†</td>
<td>0.4509†</td>
</tr>
<tr>
<td></td>
<td>Softmax</td>
<td>0.4305†</td>
<td>0.4582†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5687†</td>
<td>0.6068†</td>
</tr>
</tbody>
</table>

Figure 2: Comparing performance with different T5 model sizes. Ranking models are RankT5-EncDec fine-tuned with different losses. The 95% confidence intervals are plotted.

We do not find a consistent winner between the encoder-decoder (EncDec) and the encoder-only (Enc) model structure. A possible explanation is that the T5 decoder is less important when the model is fine-tuned for text ranking tasks with sufficient training data. **Model size comparison.** We examine how the T5 model size affects the ranking performance. We fine-tune the RankT5-EncDec model with the Softmax and the PointCE loss with different sizes of T5 model checkpoints ("Base", "Large" and "3B"). We evaluate the model performance on both data sets measured by MRR@10. Results are plotted in Figure 2.

The first observation is that the performance consistently improves when the model size increases (+7% for 3B vs. Base on NQ), highlighting the potential to enable even larger language models [6] in a similar method for better ranking performance. Another observation is that models with Softmax consistently outperform PointCE (all statistically significant with $p \leq 0.05$) and the gaps remain relatively stable across different model sizes. This might suggest that the extra benefits brought by using ranking loss cannot be compensated for by simply using larger models. **Zero-shot results.** We also compare the zero-shot performance of our ranking models fine-tuned with different ranking losses. We use a subset of BEIR [42] data sets⁴ with easily accessible corpus.⁴

---

Table 2: Zero-shot performance comparison. Ranking models are RankT5-Enc fine-tuned on the MS MARCO data set with different losses. The performance is measured by NDCG@10. The best performance for each data set is bolded.

<table>
<thead>
<tr>
<th>Data set</th>
<th>PointCE</th>
<th>Softmax</th>
<th>PointCE</th>
<th>Softmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREC-COVID</td>
<td>0.7522</td>
<td>0.8071†</td>
<td>0.4594</td>
<td>0.4401</td>
</tr>
<tr>
<td>BioASQ</td>
<td>0.5346</td>
<td>0.5635</td>
<td>0.8221</td>
<td>0.8309</td>
</tr>
<tr>
<td>NFCorpus</td>
<td>0.3263</td>
<td>0.3810</td>
<td>0.4343</td>
<td>0.4422</td>
</tr>
<tr>
<td>NQ</td>
<td>0.5959</td>
<td>0.6142</td>
<td>0.8184</td>
<td>0.8186</td>
</tr>
<tr>
<td>HotpotQA</td>
<td>0.7126</td>
<td>0.7100</td>
<td>0.6231</td>
<td>0.6254</td>
</tr>
<tr>
<td>FiQA-2018</td>
<td>0.4156</td>
<td>0.4450</td>
<td>0.8352</td>
<td>0.8516</td>
</tr>
<tr>
<td>Signal-1M</td>
<td>0.3153</td>
<td>0.3200</td>
<td>0.7493</td>
<td>0.7499</td>
</tr>
<tr>
<td>ArguAna</td>
<td>0.2252</td>
<td>0.3300</td>
<td>0.5924</td>
<td>0.5211</td>
</tr>
</tbody>
</table>

We take the RankT5-Enc models fine-tuned on the MS MARCO data set with the PointCE loss and the Softmax loss respectively, and apply them to rerank top-1000 documents returned by BM25 [39]. Table 2 summarizes ranking performance measured by NDCG@10. The ranking model fine-tuned with the Softmax loss outperforms the PointCE loss on 11 out of the 15 data sets. On average, the Softmax loss achieves more than +2.1% NDCG@10 (statistically significant with $p \leq 0.05$) which indicates that using the Softmax loss produces ranking models that generalize better to out-of-domain data. In particular, using the Softmax loss achieves larger improvement on data sets with drastically different corpus (e.g., TREC-COVID, BioASQ, NFCorpus), implying that fine-tuning the model with appropriate ranking losses can enforce the model to put less emphasis on memorization, and thus to better learn the abstract concept of "relevance" [21], regardless of what the underlying corpus is.

7 CONCLUSION

In this paper, we investigate the use of pretrained T5 models for text ranking. We propose two T5 model variants that directly outperform monoT5 Generation. A possible explanation is that the T5 decoder is less important when the model is fine-tuned for text ranking tasks with sufficient training data. We do not find a consistent winner between the encoder-decoder (EncDec) and the encoder-only (Enc) model structure. A possible explanation is that the T5 decoder is less important when the model is fine-tuned for text ranking tasks with sufficient training data. **Model size comparison.** We examine how the T5 model size affects the ranking performance. We fine-tune the RankT5-EncDec model with the Softmax and the PointCE loss with different sizes of T5 model checkpoints ("Base", "Large" and "3B"). We evaluate the model performance on both data sets measured by MRR@10. Results are plotted in Figure 2.

The first observation is that the performance consistently improves when the model size increases (+7% for 3B vs. Base on NQ), highlighting the potential to enable even larger language models [6] in a similar method for better ranking performance. Another observation is that models with Softmax consistently outperform PointCE (all statistically significant with $p \leq 0.05$) and the gaps remain relatively stable across different model sizes. This might suggest that the extra benefits brought by using ranking loss cannot be compensated for by simply using larger models. **Zero-shot results.** We also compare the zero-shot performance of our ranking models fine-tuned with different ranking losses. We use a subset of BEIR [42] data sets⁴ with easily accessible corpus.⁴

---

⁴Notice that the NQ data set in BEIR is has a different corpus and query set from the NQ data set we used earlier.


