Martin-Luther-Universität Halle-Wittenberg
Institute of Computer Science
Degree Programme Computer Science, B.Sc.

Listformer: A Listwise Learning-To-Rank Transformer

Bachelor's Thesis

Erik Reuter

1. Referee: Prof. Dr. Matthias Hagen
2. Referee: Msc. Ferdinand Schlatt

Submission date: July 31, 2022

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Halle/Saaale, July 31, 2022

.....
Erik Reuter

Abstract

The usability of search engines is heavily influenced by the quality of the ranking and the time needed to form the ranking. Therefore, learning-to-rank models have to be both efficient and effective. Especially transformer models trade effectiveness with efficiency. Pointwise transformer models are faster but not as effective as pairwise transformers. ListNet shows the potential of listwise approaches to be more effective than pairwise models. We want to use the potential and propose the first single-stage listwise transformer named Listformer. Listformer uses a global-local attention mechanism to reduce memory requirements. Our global-local attention Thus, Listformer can process a ranking of more than 20 documents without truncation in a single stage. We fine-tune Listformer on the TripClick click-log dataset using the ApproxNDCG loss. Experiments evaluating Listformer's effectiveness using the TripClick IR Benchmark show that Listformer is not as effective as our BM25 baseline yet. In exchange, for reranking 100 documents, the Listformer is slower than pointwise BERT but faster than pairwise BERT.

Contents

Chapter 1

Introduction

The ranking of documents in a search engine is crucial for usability. Users expect the most relevant documents to be presented in the first spots of the search results list. To model the relevance of a document to a query, diverse approaches have been proposed, e.g., term-weighting functions like BM25 [27] or learning to rank (LTR) models like RankNet [9] or ListNet [10]. Recent approaches to learning to rank often use a two-stage approach by first retrieving documents using a term-weighting function before reranking the retrieved documents using an LTR model [4, 7]. This done because search engines are expected to be fast and most LTR models are too slow to gather a ranking from the index directly. Three types of LTR approaches exist: pointwise, pairwise, and listwise [21].

Pointwse LTR models estimate the relevance score of a document to a query independently of other documents. In contrast, in pairwise models the relevance scores of the documents are estimated by comparing the documents in pairs. RankNet [9] has shown that pairwise models have a higher effectiveness than pointwise models. The document comparisons enrich the process of relevance prediction in pairwise LTR by allowing the model to combine information of multiple documents. We call this exchanged information inter-document information.

Listwise LTR approaches follow the intuition of creating a ranking directly from the list of search results, instead of processing the list in single or pairs of documents. The listwise LTR model ListNet [10] strengthens our assumption that inter-document information benefits LTR by having a higher effectiveness than RankNet.

In recent learning to rank approaches, transformer models were used for pointwise or pairwise learning to rank [23, 12], because of their exceptional language-modeling capabilities [14]. The reason transformer models have not widely and effectively been used for listwise learning to rank is their memory

requirements. The self-attention mechanism of transformers builds an attention score matrix quadratically increasing in size with the size of the input sequence [30]. This makes transformer models impractical to use for listwise reranking. The reranking of a list of documents needs large input sequences containing all documents from the list. To use e.g. BERT for listwise LTR with just 10 documents, one would need to either truncate the documents to around 50 tokens or need to train and predict on a GPU cluster.

We tackled the problem of quadratic memory requirements using a sparse self-attention mechanism similar to that in the Longformer transformer model [6]. The sparse self-attention brings the quadratic memory requirements of full self-attention down to linear memory requirements. Our sparse self-attention is a specialized global-local attention. Each class and separator token, and each query token are global tokens. Thus, they can attend to every other token and every token attend to them. The tokens from a document on the other hand, are local tokens and can only attend to tokens from the same document and to all global tokens. The intuition behind our global-local attention, is that a seperator token following the document's tokens is used to build a representation of the document using the documents inner information in combination of the inter-document information obtained from the other seperator tokens' document representations. With the reduction of the memory limitations, we can up-scale the input size and thus can include more documents to build a listwise LTR transformer model we named Listformer.

The other problem we tackle is the lack of order-invariance in pairwise LTR transformer models, meaning the order of documents in the input influences the relevance prediction [22], which leads to pairwise models predicting either d_i or d_j as more relevant when processing the input of a document pair (d_i, d_j). Since order-invariance is a key axiom of reranking, pairwise models like DuoBERT [22] or DuoT5 [23] are called on all permutions of document pairs (d_i, d_j) with $i \neq j$. We designed our model, especially the positional embeddings and the global-local attention, to be order-invariant. The order-invarince eliminates the bias given by the position of the document in the input. Thus, our model estimates the relevance scores of the documents based on the inter-document information rather than their position in the input.

We hypothesize that the Listformer model performs better than pairwise and pointwise models because of the language-modeling capabilities of transformers combined with the inter-document information, and the order-invariance included in the training of the listwise LTR model. To prove our hypothesis, we run experiments on the TripClick benchmark [15, 26]. We futher hypothesize, that Listformer is faster than pairwise transformer with the same number of layers. We evaluate this hypothesis by comparing the reranking time of Listformer with those of pointwise and pairwise BERT models. The rest of

the thesis is structured as follows.

We briefly overview related work in Chapter 2. In Chapter 3, we propose the Listformer model before describing the loss function and the dataset we used. The experiments, the results, and the discussion of the results are given in Chapter 4 before a conclusion is drawn, and future work is described in Chapter 5.

Chapter 2

Related Work

We first give an overview of existing listwise LTR models before outlining low-memory-footprint self-attention mechanisms and models. Our work is a combination of the listwise LTR approach and global-local attention.

Listwise Learning To Rank Mitra and Craswell describe listwise LTR as models that optimize a rank-based metric directly, rather than reducing the ranking problem to pairs of documents or single documents as in pairwise or pointwise approaches [21]. One of the first listwise methods was ListNet by Cao et al. [10]. ListNet's loss function is based on two newly introduced probabilistic functions: permutation probability and top-k probability. Permutation probability represents the likelihood of a ranking over all possible rankings. Since the number of possible rankings is of order $O(n!)$, the time complexity of permutation probability is too high, thus permutation probability is considered impractical. For a more practical approach, the authors introduced top-k probability. Top-k probability represents the probability that k documents are ranked in the top-k positions. There are $\frac{n!}{(n-k)!}$ rankings having the top-k documents in the top-k positions, which makes the top-k probability more usable. The top-k probability is used to form two probability distributions, one with the predicted ranking and one with the ranking obtained from the ground truth. The loss function is formed by applying a metric between the probability distributions, e.g., cross-entropy. ListNet shows the possibilities and challenges of listwise LTR, e.g. forming a loss function that outputs a reasonable loss for any possible ranking while having bearable time complexity.

Besides ListNet, several other listwise loss functions have been proposed, e.g., RankCosine, which represents the predicted ranking list and the ground-truth ranking list as vectors and forms a loss by applying a cosine loss function between the vectors [25]. Bruch et al. introduced ApproxNDCG [8], a loss function designed to represent the NDCG. It overcomes non-differentiabilty

of NDCG. NDCG is the computed using the DCG of the predicted ranking and the ideal DCG of the ranking. In order to compute the DCG, the rank of each document d_i with relevance score r_i needs to be determined. The rank of a document is assessed using the identity function $\mathbb{I}_{r_i > r_j}$. The identity function is 1 when $r_i > r_j$ and 0 else, and therefore is non-differentiable. The authors used an approximation by replacing the $\mathbb{I}$ with the sigmoid function: $\mathbb{I}_{r_i < r_j} \approx \sigma(r_i - r_j)$. We use this loss for our listwise LTR model, because we can optimize directly on the measure we later evaluate with.

In recent listwise LTR approaches, neural networks were used, e.g., DLCM, the group scoring function model proposed by Ai et al. [1, 2], transformer models like TABLE by Sun et al. [29] or ListBERT by Kumar and Sarkar [17]. The group scoring function is a 3-layer feed-forward neural network that does not compute relevance scores over all search results, but computes them in fixed-sized groups of search results. For example, a group size of one would form a pointwise and a group size of two a pairwise model. In order to predict the relevance of each document in the list, all permutations of the fixed group size would have to be computed using the group function. The usage of all permutations has a too high time-complexity. This problem is tackled using Monte Carlo methods, which brings the computational complexity down to $O(mn)$, making the model more practical. The group scoring function performs better than RankNet and LambdaMART.

The aforementioned TABLE [29] is trained in two stages. Firstly the underlying BERT model undergoes a type-adaptive pointwise fine-tuning phase, before a listwise loss is used for the second stage. The listwise second stage is done by predicting unnormalized relevance scores for n positive and m negative documents for each query. The resulting unnormalized relevance scores are used to compute normalized relevance scores for each positive document. Thus, the ith positive document's relevance score can be computed using the following formula:

$$score_i^+ = \frac{r_i^+}{\sum_{k=0}^{n} r_k^+ + \sum_{l=0}^{m} r_l^-}$$

The scores are then used to form the loss:

$$\mathcal{L} = \frac{-\sum_{i=0}^{n} log(score_i^+)}{n}$$

TABLE performs better than the pairwise model DuoBERT [22] on the MS MARCO dataset. The authors show in experiments that the main contribution to the model's performance is the type-adaptive fine-tuning phase. We hypothesize that a listwise model could reach higher performance by inputting all documents in one call rather than collecting the scores of each document independently. Our hypothesis is based on the assumption that a model can

learn from inter-document information when all documents can be seen in the prediction of the relevance score and not only in one linear layer.

Sparse Self-Attention Mechanisms The self-attention mechanism is the core mechanism of transformer models [30] by enabling the model to construct a contextualized token representation based on all input tokens.

The key and query vectors are multiplied to produce an attention score matrix of size n^2, which limits the input size. Therefore, most models restrict the input size to 512 tokens [14, 19]. To input larger sequences, self-attention with lower memory requirements would be needed. Approaches to tackling this problem are lightweight convolutions as attention replacements [31] or recurrent [13], hierarchical [32], compressed [18], or sparse attention mechanisms [3, 6, 11]. In a sparse attention mechanism, a token does not attend to all tokens but to a subset of tokens. With an input sequence of size n a sparse attention mechanism, where each token attends only to m tokens, the attention mechanism has a space complexity of $O(nm)$. When m is fixed, the size of the attention matrix increases linearly with the input size. For example, in the Sparse Transformer, an image generation model, pixels attend to pixels from the same row or column in the image. Thus, the space complexity of the sparse attention applied to an image with n pixels is $O(n\sqrt{n})$.

An even lower space complexity is obtained with the ETC (extended transformer construction) architecture by Ainslie et al. by using *local-global-attention* [3]. The architecture splits the input into a global input of size n_g and a long input of size n_l. The intuition behind the global input is that all tokens from the global and long input attend to the global input, and the global input attends to all tokens from the long and the global input. The long input tokens can only attend to a subset of long input tokens, but all input tokens in the global input. With global and local attention combined, linear space complexity is reached while retaining information on all tokens.

In more detail, the global-local attention is split into four pieces: global-to-global (*G2G*), global-to-local (*G2L*), local-to-global (*L2G*) and local-to-local (*L2L*). *G2G* is the part of the attention where each global input token attending to all global input token, *G2L* where each global input token attends to all long input tokens, and *L2G* where each long input token attends to all global input tokens, respectively. On the other hand, *L2L* is the part of attention where each long input token only attends to a subset of long input tokens. The attention of the global sequence is computed using *G2G* and *G2L*, and the attention of the local attention using *L2L* and *L2G*, respectively.

A very similar mechanism is used by the Longformer by Beltagy et al. [6]. The main difference is that the Longformer's inputs are an input sequence and an attention mask. The attention mask defines which tokens in the input are

considered global and which are local. According to the global-local specifications by [6], global tokens attend to all other tokens while local tokens attend to tokens within an attention window on the tokens and to all global tokens. The flexibility of the global-local attention in the Longformer allows encoding task-specific structures in the attention mask. We use the Longformer approach to global-local attention in our model and encode a list structure in the attention mask. We further adapt the local-to-local piece to the task of listwise LTR, as described in Chapter 3.

Chapter 3

Methods

In this chapter, we describe how we designed our model for listwise LTR while keeping order-invariance, to eliminate the bias given by a documents position. Therefore, we formally describe the task of listwise relevance prediction and our model's input. Then we explain the global-local attention mechanism we use. Especially, we show how we adapted our listwise LTR model to be order-invariant by explianing the document-wise local attention and document-wise positional embeddings. To fine-tune our model, we use the TripClick dataset and the ApproxNDCG loss function, both being described in the following sections.

Input Formulation We define a set of queries $Q := \{q_1, q_2, ..., q_n\}$ and a result set of documents per query $D_i := \{d_{i,1}, d_{i,2}, ..., d_{i,m}\}$, $q_i \in Q$. The sequence of tokens of query $q_i \in Q$ are denoted as q'_i, similarly the sequence of tokens of a document $d_{i,j} \in D_i$ is denoted as $d'_{i,j}$. With this notation, the input of our listwise model has the form:

$$<CLS>q'_i<SEP>d'_{i,1}<SEP>d'_{i,2}<SEP>...<SEP>d'_{i,l}<SEP>$$

where $l \leq m$ is a parameter determining the number of documents passed to the model, *CLS* is the class token, and each *SEP* token is a seperator token.

The task of listwise relevance prediction is to predict a list of relevance scores $r_{i,j}, \ldots, r_{i,l}$ for a query $q_i \in Q$ with input documents $d_{i,1}, \ldots, d_{i,l}$. The predicted relevance scores can then be used to form the ranking of the documents. With the formulation of the input and the task of listwise relevance prediction, we can describe how we designed our listwise LTR model.

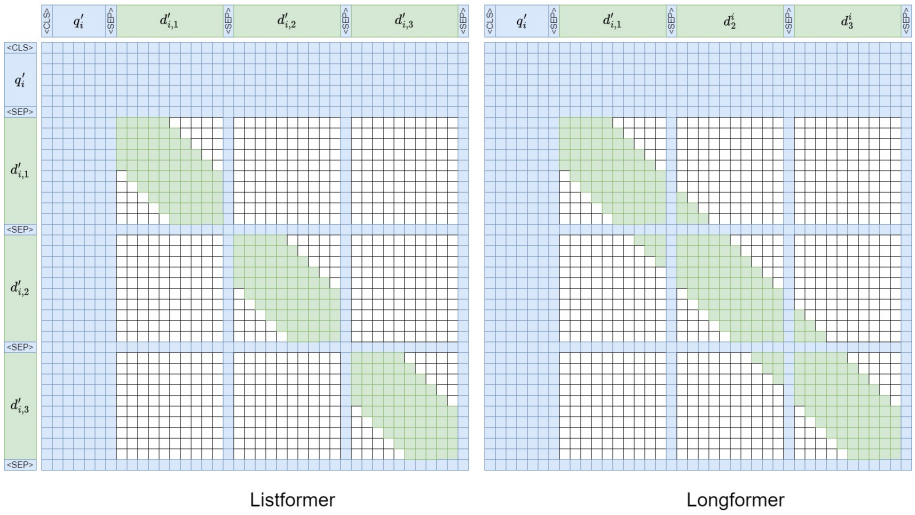

Figure 3.1: This figure shows the difference in the attention patterns of the Listformer and the Longformer model by Beltagy et al.. We use blue for tokens used in the global attention, while green tokens are used for local attention. The attention patterns show which tokens a token attends to. In both the listwise LTR transformer and the Longformer, all global tokens attend to all tokens. The differences lie in the local attention. We perform a document-wise local attention, where each token attends to tokens within an attention window of fixed size; therefore, in the Listformer, the last and first tokens of document do not attend to the last tokens of the previous, or the first tokens of the following documents.

3.1 Listformer

Our novel listwise LTR transformer model Listformer is the first model to use a single-stage approach for listwise relevance prediction. We tackle the problem of quadratic space complexity of self-attention in the original Transformer model by using a modified global-local attention mechanism. Further, we design our model to be order-invariant. Order-invariance means, that the relevance score of a document is computed purely on inner-, and inter-document information rather than on the documents position within the input. We ensure order-invariance, by designing the global-local attention mechanism to use a document-wise local attention, applying the local attention to each document individually rather than on the whole sequence. Another contribution to our listwise models order-invariance are the positional embeddings being document-wise, too. We describe our global-local attention and positional embeddings in more detail in the next paragraphs.

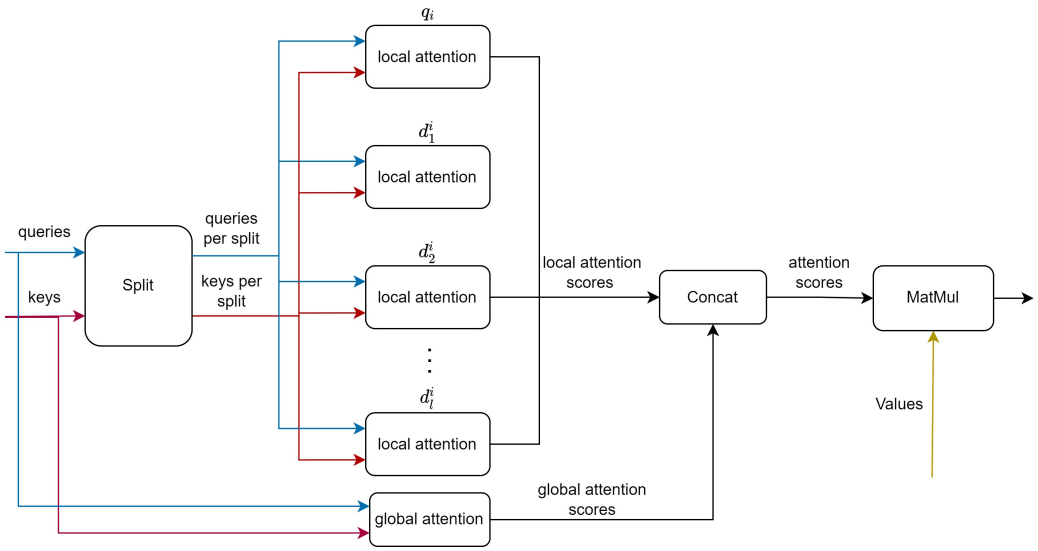

Figure 3.2: The figure depicts the document-wise global-local attention mechanism. Keys, queries and values are acquired from the hidden states of the previous transformer layer. The Listformer has two attention mechanisms, firstly the global attention, which computes a self attention over all global tokens, and the local attention. To perform the local attention, the keys and queries are devided in $l + 1$ splits, one for the query and l for the l documents passed to the model. We then perform the local attetnion on each split individually, before collecting the resulting attention scores from the local and global attention to concatenate them reobtaining the sequence-wise representation. We use the sequence-wise attention scores for multiplication with the values to obtain the attention result.

Global-Local Attention In the Longformer's local attention, tokens attend to tokens within a range around the token. Since we simply concatenate the input documents and seperate them using a *SEP* token, the last tokens of document $d_{i,j}$ would attend to the first tokens of document $d_{i,j+1}$. The document-overlapping local attention encodes an order between the documents, which contradicts the model's order-invariance. Therefore, we cut the overlaps so that the local attention is performed per document rather than sequence, as depicted in figure 3.1. In detail, we split the sequence-wise key and query vectors into document-wise key and query vectors for documents. Then a sliding attention window over the tokens of each document is used to compute the attention scores. The document-wise local attention scores then are concatenated to sequence-wise local attention scores. The global attention scores are computed by key vectors of all tokens, and the query vectors of the global tokens. The global and local attention gets concatenated and multiplied with

the value vectors. The process is depicted in Figure 3.2.

Positional Embeddings Another contribution to the order-invariance of our model are the positional embeddings. Positional embeddings encode the order of tokens in the input. When dealing with a single document, the straightforeward approach would be to assign each token its position, which is later used to produce a positional embedding. If we count up the position from the start of the input sequence to the end, we encode an order of documents, because the tokens of $d_{i,j+1}$ have higher positions than tokens of $d_{i,j}$. Therefore, we assign positions to each document's tokens independently. More precisely, each document's tokens' positions start at 0 and are counted up to the next seperator token.

Relevance Prediction To predict the relevance scores the last hidden states corresponding to the seperator token after each document are gathered, e.g., in the input $<CLS>q'_i<SEP>d'_{i,1}<\underline{SEP}>d'_{i,2}<\underline{SEP}>\ldots<\underline{SEP}>d'_{i,l}<\underline{SEP}>$ the underlined tokens will be used to gather their last hidden states. We denote those last hidden states as $s_1, s_2, ..., s_l$ where s_j corresponds to the jth seperator token. Relevance scores are predicted from $s_1, \ldots, s_l$, where W is a hidden layer projecting the output hidden states to a single value:

$$r_j = W \cdot s_j + b$$

3.2 Approximated NDCG Loss

The ApproxNDCG loss proposed by Bruch et al. [8] is designed to tackle the problem of the non-differentiability of the indicator function $\mathbb{I}$ used in the NDCG ranking metric. In more detail, the NDCG is formulated as follows:

$$\text{NDCG}(\pi_f, \boldsymbol{y}) = \frac{DCG(\pi_f, \boldsymbol{y})}{DCG(\pi^*, \boldsymbol{y})}$$

Where $\boldsymbol{y}$ is a list of relevance labels in descending order, π^* is a list of the numbers 1 to l in ascending order, and π_f is a permutation of π^*. More precisely, π_f is sorted for their predicted relevance scores $r_1, \ldots, r_l$ by model f. The DCG is formulated as

$$\text{DCG}(\pi, \boldsymbol{y}) = \sum_{i=0}^{n} \frac{2^{y_i} - 1}{\log(\pi(i) + 1)}$$

In order to obtain π_f, the indicator function $\mathbb{I}_{s<t}$ is used. The indicator function is 1 when $s > t$ and 0 else. The ith position in the ranking π_f is thus formed

Table 3.1: Statistics of the TripClick dataset.

Number of user interactions	5,272,064
Number of sessions	1,602,648
Number of unique queries	1,647,749
Number of retrieved documents	2,347,977

by:

$$\pi_f(i) = 1 + \sum_{i \neq j}^{l} \mathbb{I}_{r_i < r_j}$$

Qin et al. approximated $\mathbb{I}$ using the sigmoid function σ [24]: $\mathbb{I}_{s<t} \approx \sigma(s-t)$. Thus the approximated rank of the ith document in a list of l documents with relevance scores $r_1, \ldots, r_l$ is:

$$\widetilde{\pi}_f(i) = \text{approxRank}(i) = \sum_{i \neq j}^{l} \sigma(r_i - r_j)$$

From the approxRank function, an approxRanks function can be derived, taking in a list of relevance scores and computing the approximated rank of the relevance scores in a list.

$$\widetilde{\pi}_f = \text{approxRanks}(r_1, \ldots, r_l)$$

Now, to produce a loss function with the NDCG with the rank approximation, the NDCG needs to be negated to create a minimization problem Bruch et al.. This is done by multiplying with -1 to obtain the ApproxNDCG Loss:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = -\text{NDCG}(\text{approxRanks}(\boldsymbol{x}), \boldsymbol{y})$$

where $\boldsymbol{x}$ is the predicted list of relevance scores.

3.3 Dataset

In order to fine-tune and evaluate our model, we use the TripClick dataset, a click log dataset collected from the bio-medical search engine Trip database. TripClick wast introduced by Rekabsaz et al., it consists of around 700,000 queries, and 1.3 milion query-document relevance labels collected from 5.2 million user interactions [26]. More statistics are shown in Table 3.1.

The click logs contain user sessions consisting of queries entered by the user and the documents clicked, and retrieved per query. Rekabsaz et al. extracted

the queries and divided them into three splits for their frequency in the click logs: head queries, with a frequency higher than 44; torso queries, with a frequency between 6 and 44; and tail queries, with a frequency lower than 6. The frequencies are selected, so the head queries make up around 20%, the torso queries around 30%, and the tail queries around 50% of all queries. The splits are divided into train, test, and validation sets.

We construct samples to a query by firstly, retrieving document candidates using the pyterrier search engine [20] and BM25 [27]. Each candidate then gets annotated with a score according to a click model. Like Rekabsaz et al., we used DCTR for head queries and RAW for torso and tail queries. DCTR classifies a document's relevance into four classes using their relative click frequency. The relative click frequency of a document $d_{i,j}$ to a query q_i is defined by $rc_{i,j} = \frac{c_{i,j}}{c_i}$, where $c_{i,j}$ is the number of times $d_{i,j}$ was clicked when q_i was entered and c_i is the number of times any document was clicked. DCTR assigns a relevance label to a document $d_{i,j}$ as following: 0, when $rc_{i,j} = 0$, 1 when $0 < rc_{i,j} \leq 0.04$, 2 when $0.04 < rc_{i,j} \leq 0.3$ and 3 else. Since the documents retrieved for torso and tail queries were not clicked as often as for head queries, the binary RAW click model is more fitting. RAW assigns a relvance label of 1, when a document was click, and 0 else.

We annotate the documents using the described click models and sample from these annotated results lists using two parameters, the number of documents passed to the model $l > 0$ and a positive part in the sample $0 < p \leq 1$. The positive part specifies the part of all documents passed to the model having relevance labels higher than 0. In other words, a positive part of p would produce a sample consisting of $p \cdot l$ (relevant) documents with relevance > 0 and $(1 - p) \cdot l$ (non-relevant) documents with a relevance label of 0.

We then shuffle the sampled documents. Having a query q_i, relevant, and non-relevant documents $d_{i,1}, \ldots, d_{i,l}$, we tokenize them to get the query tokens q_i' and the document tokens $d'_{i,1}, \ldots, d'_{i,l}$. We can then form the model input by concatenating the query's and documents' tokens. Furthermore, we build an attention mask defining the type of attention per token. The attention mask entry for a token is 0 when no attention should be used, 1 when local attention should be used, and 2 if global attention should be used. Thus, a sample consists of the model input, the attention mask, and the relevance labels to the sampled documents in descending order.

We use this dataset, because of the rankings being deeply judged, e.g., the head queries have an average of 46.2 non-zero relevance judgements per query. The high number of judgements benefits the training of the Listformer, since we can train on samples having 20 or more documents with non-zero relevance labels.

Chapter 4

Experiments

In this chapter, we describe benchmarks proposed on TripClick dataset before explaining which experiments we set up to test the Listformer's effectiveness on the TripClick IR Benchmark. For the efficiency of Listformer, we test using BERT-based pointwise and pairwise rerankers as baselines for our experiments. We propose different configurations for these experiments. We then show and discuss the results of our experiments.

Benchmarks Different benchmarks have been introduced on the TripClick dataset so far. Rekabsaz et al. set up baselines using BM25 and machine learning models, e.g., KNRM (kernel-based neural ranking model), and MP (match pyramid). The authors used a negative sampling strategy, where all non-clicked documents are considered non-relevant. This strategy will likely produce false negative samples because relevant documents in the first few positions in the search results may not have been clicked.

Hofstätter et al. uses a more sophisticated negative sampling strategy by considering non-clicked search results with a low BM25 score for negative sampling. The authors used this strategy to train BERT-style transformer models to establish a benchmark on TripClick. In more detail, the authors used pre-trained models, e.g. SciBERT [5], DistilBERT [28], or Colbert [16], and fine-tuned them on the head, torso, and tail splits, individually.

We use the TripClick benchmark by [15] to evaluate the Listformer since the BERT-style transformers are more comparable to our listwise LTR transformer than the models used by Rekabsaz et al..

To create a benchmark for efficiency, we used the task of reranking 100 documents and measured the time used for reranking. The benchmark model we use is BERT [14] for pointwise and pairwise reranking. We measure how much time BERT needs to predict relevance scores for the documents in a pointwise and in a pairwise manner. For the pointwise BERT, we simulate

pointwise reranking by passing 100 inputs, each consisting of 512 random tokens to BERT. The reranking time starts when the first input is passed and ends when all 100 inputs are processed. We approximate the reranking time needed to rerank the documents using pairwise BERT by taking pointwise BERT's reranking time. Since pairwise BERT is called $n(n-1)$ times for a list of n documents, and we know how much reranking time BERT needs to process n inputs, we can approximate the reranking time of pairwise BERT by $t_{\text{pointwise}}(n) \cdot (n-1)$, where $t_{\text{pointwise}}(n)$ is pointwise BERT's reranking time for n documents.

4.1 Configurations

In this section, we explain how we fine-tune our model before describing the different model settings we experimented with.

Fine-Tuning We use a pre-trained Longformer model[1] as bases to fine-tune our model. The fine-tuning is done using the train split of the head queries. We sample these queries as described in section 3.3. The model inputs of a sample are passed to our listwise LTR model to predict relevance scores. The ApproxNDCG loss is computed from the predicted scores and the labels before gradient descent is used to optimize the model. To fine-tune our model with different configurations, we trained for $5,000$ steps with a learning rate of $1 \cdot e^{-5}$, a warm-up of 100 steps, and a weight-decay of 0.01. For the local attention we used an attention window of size 512. We trained with a batch size of 4 samples on 4 Nvidia V100 GPUs having 40 GB of memory each.

Experimental Setup To evaluate the effectiveness of the Listformer, we used the fine-tuned configurations to rerank the first 100 documents retrieved by BM25. This is done by building an input out of the query and the 100 top-documents and passing it to the model to rerank the documents for the predicted relevance scores.

We build a single input for the efficiency benchmark by concatenating a query and 100 documents separated by *SEP* tokens. The query and the documents consist of 512 random tokens each. The input is passed to the Listformer. The time needed for reranking starts when the input is passed and ends when the relevance predictions are received.

[1] `https://huggingface.co/allenai/longformer-base-4096`

Table 4.1: Listed are the different configurations we experimented with. We experimented with a base configuration of 20 documents and a positive part of 0.5 and derived a configuration with 40 documents and 50% positive documents. Due to the poor effectiveness of both configurations, we did not build further configurations.

Configuration	Number of Documents	Positive Part
$\text{Listformer}_{(20,0.5)}$	20	0.5
$\text{Listformer}_{(40,0.5)}$	40	0.5

Model Settings In our listwise LTR model, we can configure the number of documents per sample and the positive part in a sample. Our base configuration uses 20 documents per sample with 50% of them being positive. Derived from this configuration we set up configurations with a higher number of documents. Due to the poor performance of our base configuration, we did not experiment with the number of documents or the positive part further.

We listed the configurations in Table 4.1.

4.2 Results

The results of our experiments for effectiveness show, that all Listformer configurations have a much lower effectiveness than the BM25 baseline and is far behind other transformer models. This suggests that Listformer has major issues creating a ranking. We suspect that we either made mistakes in the implementation of the Listformer, or in the design of the global-local attention.

The results of our efficiency experiment proof the hypothesis, that Listformer has a higher efficiency than pairwise transformer models using a similar number of layers. We further see that pointwise BERT is much faster, suggesting that Listformer's efficiency can be further optimized.

Our results for experiment on effectiveness is shown in Table 4.2, and the results for the efficiency experiment in Table 4.3.

Table 4.2: Experimental results on of the TripClick IR Benchmark. We used NDCG scores at ranks 5, 10, 20, and 100 to evaluate the effectiveness of the Listformer. The NDCG scores of $BM25_{benchmark}$ and $BERT_{CAT}$ were taken from the TripCLick IR Benchmark.

Model Name	NDCG@5	NDCG@10	NDCG@20	NDCG@100
Head (DCTR)				
$Listformer_{(20,0.5)}$	0.086	0.102	0.146	0.236
$Listformer_{(40,0.5)}$	0.089	0.104	0.147	0.245
$BM25_{pyterrier}$	**0.123**	0.139	**0.167**	**0.256**
$BM25_{benchmark}$	-	0.140	-	-
$BERT_{CAT}$	-	**0.303**	-	-
Head (RAW)				
$Listformer_{(20,0.5)}$	0.111	0.110	0.119	0.255
$Listformer_{(40,0.5)}$	0.110	0.110	0.120	0.253
$BM25_{pyterrier}$	**0.199**	0.197	**0.209**	**0.307**
$BM25_{benchmark}$	-	0.199	-	-
$BERT_{CAT}$	-	**0.409**	-	-
Torso (RAW)				
$Listformer_{(20,0.5)}$	0.042	0.054	0.084	0.226
$Listformer_{(40,0.5)}$	0.040	0.056	0.080	0.223
$BM25_{pyterrier}$	**0.183**	0.210	**0.253**	**0.338**
$BM25_{benchmark}$	-	0.206	-	-
$BERT_{CAT}$	-	**0.370**	-	-
Tail (RAW)				
$Listformer_{(20,0.5)}$	0.026	0.038	0.057	0.172
$Listformer_{(40,0.5)}$	0.025	0.041	0.062	0.181
$BM25_{pyterrier}$	**0.234**	0.269	**0.301**	**0.348**
$BM25_{benchmark}$	-	0.267	-	-
$BERT_{CAT}$	-	**0.420**	-	-

Table 4.3: Efficiency of the Listformer in comparison to a pointwise and a pairwise BERT, where each model is used to predict relevance scores of 100 documents to a query.

Model Name	time in s
Listformer	8.32
$BERT_{(}$pointwise)	0.13
$BERT_{(}$pairwise)	12.57

Chapter 5

Conclusion

We propose a listwise transformer model named Listformer. Listformer uses a global-local attention, similar to the attention used in the Longformer model [6]. The global-local attention mechanism decreases the otherwise quadratic memory requirements of self-attention used in the original transformer [30] to linear memory requirements. The lower memory requirements open the possibility of single-stage listwise LTR, since the input length can be higher than the restrictions used in common transformer model. Hence, whole lists of search results can be passed in the model. We designed our model to be order-invariant, meaning the oder of the documents in the input does not affect the relevance prediction. The order-invariance removes the bias given by a documents position in the input.

We fine-tuned our model on the bio-medical domain using the TripClick dataset and evaluated the Listformer on the TripClick IR Benchmark for its effectiveness. The results show that the Listformer has a lower effectiveness than the BM25 baseline. Further, we evaluated the Listformer's efficiency by comparing the reranking time of Listformer with those of pointwise and pairwise BERT models. We found out, that Listformer is faster than pairwise BERT and slower than pointwise BERT. Which shows that listwise transformer models can make use of inter-document information while having a lower runtime than pairwise models.

Future Work The next steps will be to fix our model to get reasonable results. When we have model with a decent performance we will create more configurations for experiments. For example, we can experiment with the attention window size or with different global-local attention mechanisms.

In order to reach higher efficiency, we want to train a distilled version of the Listformer.

Bibliography

[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 135–144. https://doi.org/10.1145/3209978.3209985

[2] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2019. Learning Groupwise Multivariate Scoring Functions Using Deep Neural Networks. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. 85–92. https://doi.org/10.1145/3341981.3344218 arXiv:1811.04415 [cs].

[3] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. *ETC: Encoding Long and Structured Inputs in Transformers*. Technical Report arXiv:2004.08483. arXiv. https://doi.org/10.48550/arXiv.2004.08483 arXiv:2004.08483 [cs, stat] type: article.

[4] Nima Asadi and Jimmy Lin. 2013. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, Dublin Ireland, 997–1000. https://doi.org/10.1145/2484028.2484132

[5] Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3615–3620. https://doi.org/10.18653/v1/D19-1371

[6] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. *Longformer: The Long-Document Transformer*. Technical Report arXiv:2004.05150. arXiv. `https://doi.org/10.48550/arXiv.2004.05150` arXiv:2004.05150 [cs] type: article.

[7] Leonid Boytsov and Eric Nyberg. 2020. Flexible retrieval with NMSLIB and FlexNeuART. (Nov. 2020). `https://doi.org/10.48550/arXiv.2010.14848` arXiv:2010.14848 [cs].

[8] Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. 2019. Revisiting Approximate Metric Optimization in the Age of Deep Neural Networks. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Paris France, 1241–1244. `https://doi.org/10.1145/3331184.3331347`

[9] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning (ICML '05)*. Association for Computing Machinery, New York, NY, USA, 89–96. `https://doi.org/10.1145/1102351.1102363`

[10] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning - ICML '07*. ACM Press, Corvalis, Oregon, 129–136. `https://doi.org/10.1145/1273496.1273513`

[11] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. (April 2019). `https://doi.org/10.48550/arXiv.1904.10509` arXiv:1904.10509 [cs, stat].

[12] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. (Feb. 2021). `https://doi.org/10.48550/arXiv.2102.07662` arXiv:2102.07662 [cs].

[13] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2978–2988. `https://doi.org/10.18653/v1/P19-1285`

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Technical Report arXiv:1810.04805. arXiv. `https://doi.org/10.48550/arXiv.1810.04805` arXiv:1810.04805 [cs] type: article.

[15] Sebastian Hofstätter, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. *Establishing Strong Baselines for TripClick Health Retrieval*. Technical Report arXiv:2201.00365. arXiv. `https://doi.org/10.48550/arXiv.2201.00365` arXiv:2201.00365 [cs] type: article.

[16] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Virtual Event China, 39–48. `https://doi.org/10.1145/3397271.3401075`

[17] Lakshya Kumar and Sagnik Sarkar. 2022. ListBERT: Learning to Rank E-commerce products with Listwise BERT. (June 2022). `https://doi.org/10.48550/arXiv.2206.15198` arXiv:2206.15198 [cs].

[18] Jiayi Li and Yujiu Yang. 2022. *STaR: Knowledge Graph Embedding by Scaling, Translation and Rotation*. Technical Report arXiv:2202.07130. arXiv. `https://doi.org/10.48550/arXiv.2202.07130` arXiv:2202.07130 [cs] type: article.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. Technical Report arXiv:1907.11692. arXiv. `https://doi.org/10.48550/arXiv.1907.11692` arXiv:1907.11692 [cs] type: article.

[20] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 4526–4533. `https://doi.org/10.1145/3459637.3482013`

[21] Bhaskar Mitra and Nick Craswell. 2018. An Introduction to Neural Information Retrieval. *Foundations and Trends® in Information Retrieval* 13, 1 (Dec. 2018), 1–126. `https://doi.org/10.1561/1500000061` Publisher: Now Publishers, Inc.

[22] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-Stage Document Ranking with BERT. (Oct. 2019). `http://arxiv.org/abs/1910.14424` arXiv:1910.14424 [cs].

[23] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. (Jan. 2021). `https://doi.org/10.48550/arXiv.2101.05667` arXiv:2101.05667 [cs].

[24] Tao Qin, Tie-Yan Liu, and Hang Li. 2008. A General Approximation Framework for Direct Optimization of Information Retrieval Measures. (Nov. 2008). `https://www.microsoft.com/en-us/research/publication/a-general-approximation-framework-for-direct-optimization-of-information-ret`

[25] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. 2008. Query-level loss functions for information retrieval. *Information Processing & Management* 44, 2 (March 2008), 838–855. `https://doi.org/10.1016/j.ipm.2007.07.016`

[26] Navid Rekabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2021. TripClick: The Log Files of a Large Health Web Search Engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2507–2513. `https://doi.org/10.1145/3404835.3463242` arXiv:2103.07901 [cs].

[27] S. Robertson, S. Walker, Susan Jones, M. Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *TREC*.

[28] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. Technical Report arXiv:1910.01108. arXiv. `https://doi.org/10.48550/arXiv.1910.01108` arXiv:1910.01108 [cs] type: article.

[29] 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)*. Association for Computing Machinery, New York, NY, USA, 2233–2236. `https://doi.org/10.1145/3340531.3412071`

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. *Attention Is All You Need*. Technical Report arXiv:1706.03762. arXiv.

https://doi.org/10.48550/arXiv.1706.03762 arXiv:1706.03762 [cs] type: article.

[31] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. *Pay Less Attention with Lightweight and Dynamic Convolutions*. Technical Report arXiv:1901.10430. arXiv. https://doi.org/10.48550/arXiv.1901.10430 arXiv:1901.10430 [cs] type: article.

[32] Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: Document Level Pre-training of Hierarchical Bidirectional Transformers for Document Summarization. (May 2019). https://doi.org/10.48550/arXiv.1905.06566 arXiv:1905.06566 [cs].