EARL: Speedup Transformer-based Rankers with Pre-computed Representation

Luyu Gao **Zhuyun Dai** **Jamie Callan**
Language Technologies Institute
Carnegie Mellon University
{luyug, zhuyund, callan}@cs.cmu.edu

Abstract

Recent innovations in Transformer-based ranking models have advanced the state-of-the-art in information retrieval. However, their performance gains come at a steep computational cost. This paper presents a novel Embed Ahead Rank Later (EARL) framework, which speeds-up Transformer-based rankers by pre-computing representations and keeping online computation shallow. EARL dis-entangles the attention in a typical Transformer-based ranker into three asynchronous tasks and assign each to a dedicated Transformer: query understanding, document understanding, and relevance judging. With such a ranking framework, query and document token representations can be offline computed and reused. We also propose a new judger transformer block that keeps online relevance judging light and shallow. Our experiments demonstrate that EARL can be as effective as previous state-of-the-art BERT rankers in accuracy while substantially faster in evaluation time.

1 Introduction

Recent innovations in neural rankering models have advanced the state-of-the-art retrieval performance (Xiong et al., 2017a; Dai et al., 2018; Nogueira and Cho, 2019a; Dai and Callan, 2019b). Neural rankers based on Transformer architectures (Vaswani et al., 2017) fine-tuned from BERT (Devlin et al., 2019) achieve current state-of-the-art ranking accuracy (Nogueira and Cho, 2019a; Dai and Callan, 2019b). The power of Transformer comes from attention, the process by which it interacts all possible pairs of words within the context window to understand the connections between them. When used for ranking, a pair of query and document are concatenated together and fed into the Transformer, where the attention mechanism build up contextualized token representations and query-document token interactions. Attention provides detailed, token-level information for matching, which has been proven critical to the effectiveness of Transformer-based rankers (Wu et al., 2019).

However, the performance gains of Transformer-based rankers come at a steep computational cost. The attention computation scales quadratically to the input length (Vaswani et al., 2017). While queries are normally short, the documents can be long and, therefore, the concatenated input. Moreover, the total computation also scales with number of transfomer blocks used and previous state-of-th-art BERT rankers are all very deep with twelve of them. On the other hand, ad-hoc retrieval needs to process each query within a short amount of time. As a result, transformer-based rankers that take long to score a single query-document pair needs to adopt a shorter candidate list to rerank, which may lead to substantial drops in final reranking effectiveness (Dai and Callan, 2019a). With a defined transformer model, one could only trade-off between response time and ranking performance. It remains an open question how to reduce the computational cost of Transformer-based rankers and still retain the detailed token-level attention signals that are critical to the effectiveness.

This paper aims to address these challenges with the Embed Ahead Rank Later framework (EARL), a novel ranking framework designed to speedup Transformer-based rankers by pre-computing representations and keeping online computation shallow. EARL dis-entangles the attention in a typical Transformer ranker into three separate tasks and assign each to a dedicated Transformer module: document understanding, query understanding, and relevance judging modules. The **document understanding** module is a Transformer that takes in documents and builds contextualized document token representations beforehand in a query agnostic manner, i.e., the documents are embedded *ahead of time*. The **query understanding** module is another Transformer that creates contextualized query token representations when the initial ranker is building the candidate list of documents for reranking. We recognize the importance of token-level information for matching and never collapse tokens with pooling in the query/document understanding modules. The **relevance judging** module looks up the pre-computed document and query token representations, and use a third transformer to model query-document interactions to produce a relevance estimation.

With such a ranking framework, document and query representations can be pre-computed and reused, therefore reducing online computation cost. In EARL, all of document understanding modules work is shifted offline, query understanding runs only once when receiving the query, and only the relevance judging model needs to run through every query-document pair. To further accelerate the relevance judging module, we limit the entire relevance judging module to be a shallow transformer. Furthermore, we introduce a novel light cross-attention technique that only attends from query to document, avoiding self-attention over document tokens at all. Additionally, we propose two document representation reuse strategies for EARL to help system developers trade-off between time and space.

We test the effectiveness and efficiency of EARL on two widely-used ad hoc retrieval datasets. Our experiments demonstrates that EARL models can be as effective as previous state-of-the-art BERT-based rankers while keeping the online computation low using a light and shallow relevance judging module. Both our mathematical analysis and empirical speed measurement demonstrate EARL's superior efficiency.

2 Related Work

Neural ranking models for IR proposed in previous studies can be generally classified into two groups: *representation-based* models , and *interaction-based* models.

Representation-based models learn latent vectors (embeddings) of queries and documents and use a simple scoring function (e.g., cosine) to measure the relevance between them. Such methods can dates back to the 1990s, including LSI (Deerwester et al., 1990), earlier neural vector space models like MatchPlus (Caid et al., 1995), and classical siamese networks (Bromley et al., 1993). More recent research considered using modern deep learning techniques to learn the representations. Example include DSSM (Huang et al., 2013), C-DSSM (Shen et al., 2014), etc. Representations-based models are very efficient during evaluation because the document representations are independent of the query, and therefore can be pre-computed. However, representing documents with a single low-dimensional vector lose specific term matching signals, which are critical to IR (Salton and McGill, 1984). As a result, previous representation-based ranking models mostly fail to outperform interact-based ones. Recently, Reimers and Gurevych (2019) proposed a new representation-based model that uses BERT as the encoder. The proposed model was tested on several sentence similarity tasks, but its effectiveness for ad hoc search remains to be studied.

Interaction-based models, on other hands, use a neural network to model the word-level interactions between the query and the document. For example, ARC-II (Hu et al., 2014) build hierarchical Convolutional Neural Networks (CNN) on the interactions of two texts word embeddings; K-NRM (Xiong et al., 2017b) and Conv-KNRM (Dai et al., 2018) uses Gaussian kernel pooling to summarize the word-level similarities into ranking signals. Recently, transformers (Vaswani et al., 2017), especially BERT (Devlin et al., 2019) based transformers, have been widely used in the search ranking task (Nogueira and Cho, 2019b; Dai and Callan, 2019b). These BERT-based rankers concatenate query and document into a single string, and apply self-attention that spans over the query and the document in every layer. This yields a rich set of interaction signals from every possible pair of words in the concatenated query-document string. Rankers using pre-trained transformers such as BERT has become the current state-of-the-art (Nogueira and Cho, 2019b; Craswell et al., 2019). However, the performance gains come at the computational cost of inferring the many token-level interaction signals at the evaluation time, which scales quadratically to the input length.

Comparing the two categories of neural rankers mentioned above, we can see that representation-based models have higher efficiency with pre-computed representations. In contrast, interaction-based models excel in accuracy by modeling the word-level interaction signals. A possible direction is to combine the advantages of both worlds. However, little prior research has studied this direction. It is an open question whether we can accelerate state-of-the-art rankers by pre-computing certain representations and also retaining the specific word-level interaction signals.

There are several research directions aiming to reduce the computational cost of Transformer models. One line of research seeks to compress the big transformer into smaller ones using model pruning (Voita et al., 2019) or knowledge distillation (Hinton et al., 2015; Sanh et al., 2019). Another line of research aims to develop new transformer-like units that have lower complexity than the original transformer. For example, (Child et al., 2019) introduces sparse factorizations of the attention matrix which efficiently compute subsets of the attention matrix. The focus of our work is a ranking framework that pre-computes and re-uses representations; faster Transformer models are orthogonal to this paper.

3 Proposed Method

In this section, we introduce the Embed Ahead Rank Later (EARL) ranking framework, how EARL speeds-up ranking through representation pre-computing, and how EARL is trained.

3.1 Notations

Following notations in (Vaswani et al., 2017), we write attention among three matrices X, Y, Z as Attention(X, Y, Z), i.e.

$$\text{Attention}(X, Y, Z) = softmax(\frac{XY^T}{\sqrt{d_{model}}})Z \tag{1}$$

Multi-head attention among X, Y and Z matrices is denoted as MH(X, Y, Z), i.e.,

$$\text{MH}(X, Y, Z) = \text{MH}(head_1, ..., head_h)W^O \tag{2}$$

$$head_i = \text{Attention}(XW_X^i, YW_Y^i, ZW_Z^i) \tag{3}$$

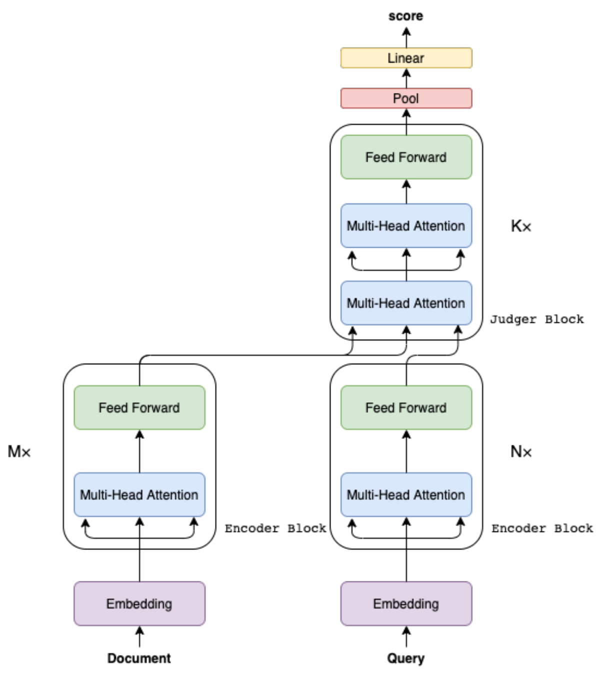


Figure 1: EARL Model Architecture. For simplicity, we omit residual connections and layer normalization from the illustration.

The position-wise feed-forward neural network is denoted as,

$$\text{FFN}(x) = ReLU(xW1 + b1)W2 + b2 \tag{4}$$

Layer normalization function with input x is written as Norm(x).

With these notations, we will formally describe our EARL framework in the next section.

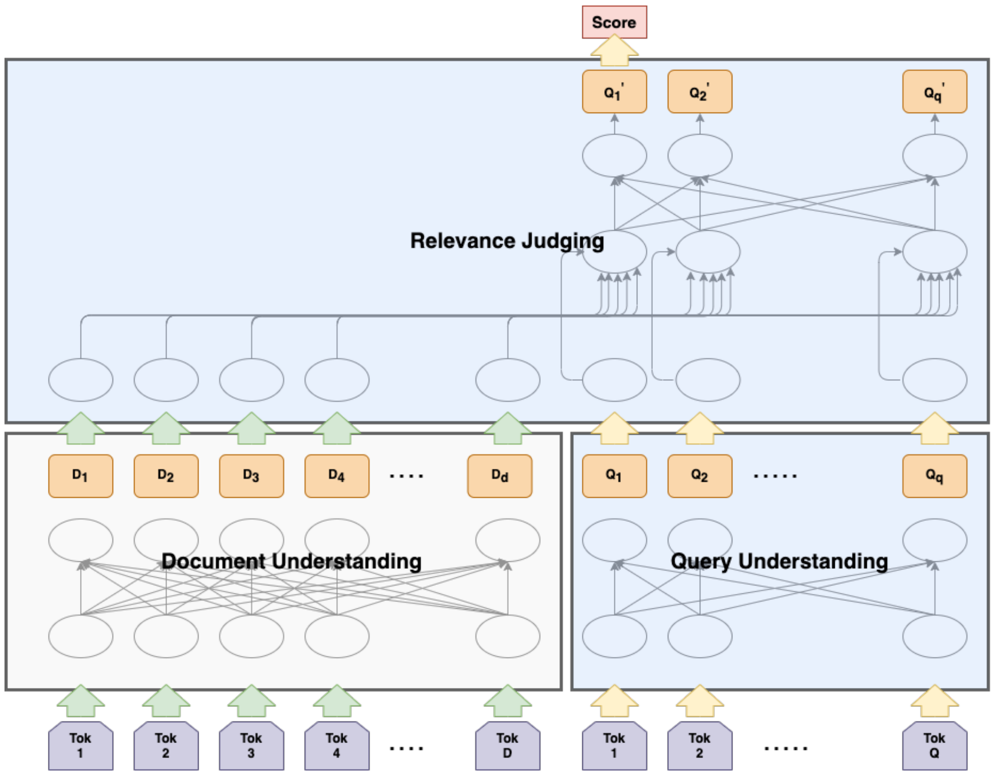
3.2 The EARL Framework

A typical Transformer-based ranker (Nogueira and Cho, 2019b; Dai and Callan, 2019b; Wu et al., 2019) takes in the concatenation of a query qry and a document doc as its input, and produces a relevance score:

$$score(qry, doc) = f(T(concat(qry, doc))) \tag{5}$$

where T is a Transformer, and f is a scoring function defined on the output of the Transformer. At each layer, the Transformer generates a new contextualized embedding for each token based on its attention to all tokens in the concatenated text. This step runs in quadratic time to the input length and must be computed during query evaluation time. As shown in Table 1, for a query of q tokens and a document of d tokens, the Transformer would require assessments of $(q + d)^2$ pairs of tokens .

Figure 2: An illustration of the attention within EARL's modules. The illustration demonstrates attention flows within EARL and does not reflect actual model depth. In a real model, understanding modules are deeper than shown here.



EARL aims to dis-entangle the attention and pre-computes a large portion of them in advance. Figure 1 gives an overview of the EARL framework. EARL divides up the relevance ranking task into three parts and assign each to a dedicated Transformer module: document understanding T_{doc}, query understanding T_{qry}, and relevance judging T_{judge}. To score a query-document pair, the document understanding module and the query understanding module first model the query/document context separately and build the corresponding query and document token representations,

$$D = T_{doc}(doc) \tag{6}$$

$$Q = T_{qry}(qry) \tag{7}$$

Then the relevance judging module looks up these pre-computed representations and model the attention between query and document token:

$$score(qry, doc) = T_{judge}(Q, D) \tag{8}$$

Consequently, for all documents in a corpus, T_{doc} can run off-line, while T_{qry} runs only the first time a query comes in and the result can be reused afterwards, e.g. when scoring second document in a candidate list of a query.

Document Understanding Module. The document understanding module T_{doc} builds document token-level representations using a Transformer model. It takes in only the document (*without seeing the query*) and outputs a query agnostic document representation. As a result, it only needs to be run over the corpus once, which can be done off-line with results cached.

Figure 2 illustrate the detailed attention in EARL. As shown in the figure, the document understanding module is the same as a transformer encoder (Vaswani et al., 2017). For an input document of length d, we first embed it with document embedding look-up layer $embed_{doc}$ to get a word-level embedding matrix $E_{doc}^{d \times n}$,

$$E_{doc} = embed_{doc}(document) \tag{9}$$

Following (Devlin et al., 2019), we prepend a [CLS] token to the document text for sentence-level information aggregation. These document token embeddings are then processed with a series of M transformer encoder blocks, i.e, to produce a series of hidden representations $H_1^{doc}, ..H_M^{doc}$

where,

$$H_l^{doc} = \text{Encoder}_l^{doc}(H_{l-1}^{doc}) \tag{10}$$

$$H_1^{doc} = \text{Encoder}_1^{doc}(E_{doc}) \tag{11}$$

Each encoder block is distinct, defined following (Vaswani et al., 2017) as,

$$A^l = \text{Norm}(\text{MH}(H_{l-1}^{doc}, H_{l-1}^{doc}, H_{l-1}^{doc}) + H_{l-1}^{doc}) \tag{12}$$

$$\text{Encoder}_l^{doc}(H_{l-1}^{doc}) = \text{Norm}(\text{FFN}(A^l) + A^l) \tag{13}$$

We use the output from the last layer of the transformer as the final document representation, denoted as $D = H_M^{doc}$. D is a sequence of d contextualized token embeddings for a document of with d tokens, where each embedding is of size n. Unlike many previous representation-based neural rankers (Huang et al., 2013), we do not apply any pooling over the token embeddings. In this way, the document representation retains granular information for the later relevance judging.

Query Understanding Module. On the query side, we use a transformer encoder T_{qry} to process the input query independent of documents. The result is a token-level query representation that needs to be computed only once and can be used for multiple documents at the ranking time.

Similar to document understanding module, we first embed input query of length q with an query word look-up embedding layer, into look-uped query embedding $E_{qry}^{q\times n}$

$$E_{qry} = embed_{qry}(query) \tag{14}$$

We also prepend ¡CLS¿ token to aggregate query sentence level information. A series of N transformer encoder blocks are used to process the token embeddings into a set of hidden query representations, $H_1^{qry}, ..H_N^{qry}$,

$$H_l^{qry} = \text{Encoder}_l^{qry}(H_{l-1}^{q}) \tag{15}$$

$$H_1^{qry} = \text{Encoder}_1^{qry}(E_{qry}) \tag{16}$$

We use the output from the last layer of the transfomer T_{qry} as the final query representation, denoted as $Q = H_N^{qry}$. Q is of size $q \times n$, consisting of a sequence of q contextualized token embeddings for a query of length q.

Relevance Judging Module. The relevance judging module T_{judge} is used to judge the relevance between document and query based on their representations independently generated by the previous two modules. For a query-document pair, it takes in the query representation Q, looks up the pre-computed document representation D, and models the query-document interactions using attention.

If we use a standard Transformer (Vaswani et al., 2017), the relevance judging module would still need to assess the $(q + d)^2$ pairs of token embedding interactions, which is slow. We address this challenge with a new type of transformer block, the *judger block*. The judger block uses only the query tokens as attention anchors and avoids both self-attention among document tokens and attention from document tokens to query tokens.

As shown in Figure 2, the judger block applies a query-to-document cross-attention, followed by a query-side self attention:

$$Q_{\text{cross}} = \text{Norm}(\text{MH}(Q, D, D) + Q) \tag{17}$$

$$Q_{\text{self}} = \text{Norm}(\text{MH}(Q_{\text{cross}}, Q_{\text{cross}}, Q_{\text{cross}}) + Q_{\text{cross}}) \tag{18}$$

Eq.17 models interactions from query tokens to document token. Each query token in Q serves as an anchor and attend to document tokens in D to produce relevance signals. On the other hand, Eq.18 collects and exchanges signals from different query tokens by having all query tokens attending to each other. As shown in Table 1, the cross-attention considers qd attention pairs, and the self-attention considers q^2 attention pairs. It avoids the computational-heavy d^2 document attention pairs computed in the vanilla Transformer.

The updated query token representations are then passed to a feed-forward network on the query token embeddings with residual connections

$$H_1^j = \text{Judger}(Q, D) = \text{Norm}(\text{FFN}(Q_{\text{self}})) + Q_{\text{self}}) \tag{19}$$

We employ multiple layers of judger blocks to iteratively repeat this process and refine the hidden query token representations, producing a series of hidden judger states $H_1^j, ..H_K^j$, i.e.

$$H_l^j = \text{Judger}_l^j(H_{l-1}^j, D) \tag{20}$$

$$H_1^j = \text{Judeger}_1^j(Q, D) \tag{21}$$

In the experiment session, we will show that we can keep the layers of judger blocks small in adhoc retrieval tasks.

Table 1: Time complexity of EARL and a standard BERT ranker. We write q for query length, d for document length, n for Transformer's hidden layer dimension, and N_{doc} for number of candidate documents to be ranked for each query.

Method	total, 1 query-document pair	online, 1 query-document pair	online, N_{doc} documents
BERT ranker	$n(d+q)^2$ $+n^2(d+q)$	$n(d+q)^2+n^2(d+q)$	$(n(d+q)^2+n^2(d+q))N_{doc}$
Document Understanding	nd^2+n^2d	0	0
Query Understanding	nq^2+n^2q	nq^2+n^2q	(nq^2+n^2q)
Relevance Judghing (S1)	$n(qd+q^2)+n^2(q+d)$	$n(qd+q^2)+n^2(q+d)$	$(n(qd+q^2)+n^2(q+d))N_{doc}$
Relevance Judghing (S2)	$n(qd+q^2)+n^2(q+d)$	$n(qd+q^2)+n^2q$	$(n(qd+q^2)+n^2q)N_{doc}$

To induce relevance, we pool the last layer's query token representation and linearly project it into a score,

$$score(qry, doc) = \mathbf{w}^T \text{Pool}(H_K^j) \tag{22}$$

We experimented with mean and pooling and pooling with first [CLS] token in the relevance judging module.

The entire relevance judging module can be viewed as a transformer-based learnable metric function between query and document token-level representations. The proposed judger block modifies the standard Transformer for further speedup. It avoids performing document token to document or query token attention, which is often expensive due to long document length. Moreover, such an attention mechanism only updates query token representations but keeps the document token representations *unchanged*. As will shown in Section 3.3, keeping document representations static allows CLEAR to partially pre-compute intermediate results in the relevance judging module, which leads to further speedup.

3.3 Representation Pre-Computing and Reusing

This section details the pre-computation and reuse strategies in EARL, and discusses how they reduce the online computation complexity.

The query understanding module runs once when receiving the new query; the query token representations are then repeatedly used to rank the candidate documents. The document representations are built offline without seeing the query. We detail two representation reuse strategies with different time vs. space trade-offs: 1) a document representation reuse strategy that stores the document understanding module's output, and 2) a transformed document representation reuse strategy that stores the relevance judging module's intermediate transformed document representations.

Document representation reuse strategy (S1) pre-computes and stores D, the output of the document understanding module as described in 3.2. When receiving a new query, EARL will look-up document representations D for candidate documents, runs query understanding module to get a query's representation Q, and feed both to the relevance judging module for relevance judgement. In essence, this strategy leads to compute being saved by not running the document understanding module at query time.

Transformed document representation reuse strategy (S2) further seeks to save document-related computation performed within the relevance judging module. Recall that within each judger block of the relevance judging module, a multi-head cross attention is performed from query representations to document representations. To perform the attention, the document token embeddings need to be transformed into the Transformer's key and value spaces through linear projections.

$$\text{MH}(H_i^j, D, D) = \text{MH}(head_1, ..., head_h)W^O \tag{23}$$

$$head_n = \text{Attention}(H_i^j W_X^n, DW_Y^n, DW_Z^n) \tag{24}$$

where H_i^j is the contextualized query token representations, D is the *pre-computed* document token representations produced by the document understanding module, and W_X^n, W_Y^n, W_Z^n are head n's transformation matrices. Note that DW_Y^n and DW_Z^n are document only component. This formulation allows us to pre-compute these document transformations ahead of time, i.e., for cross attention with h heads, we pre-compute and store D':

$$D' = \{(DW_Y^n, DW_Z^n)\}_{n=1}^h. \tag{25}$$

Compared to Strategy 1, Strategy 2 moves document-only computation in the relevance judging module into the offline process. The relevance judging module will use online computed Q and the saved transformed document representation D' to make relevance judgment.

In spirit, the two strategies are similar in the sense that both avoid computation over the document side by pre-computing and reusing document represented. The main difference is that the second strategy pre-computes document transformations into multiple embedding spaces. With this extra pre-computation, strategy 2 trades storage for a further speed-up.

Table 1 analyzes the online time complexity of EARL, and compares it to the time complexity of a standard BERT ranker. We note that EARL moves all document only computation to offline. It totally avoids the quadratic term d^2, which are often the most expensive part due to long document length. The reusing strategy two (S2) further removes the document transformation term n^2d, one that is linear in document length and quadratic in model dimension, from online computation.

3.4 Training EARL

EARL needs to learn three Transformers: T_{doc} for document understanding, T_{qry} for query understanding, and T_{judge} for relevance judging. The three Transformer modules are *coupled* during training and *decoupled* when using. To train EARL, we connect the three Transformers and enforce module coupling by training end-to-end using standard ranking loss functions such as the pairwise hinge loss (Nogueira and Cho, 2019b; Dai and Callan, 2019b). When training is finished, we decouple the three Transformer modules and apply each module at the desired offline/online time.

We were not able to pre-train EARL due to hardware constraints. However, we would still like to use pre-trained language model weight to ease optimization and give EARL better language understanding ability. To do so, we propose a initialization scheme for EARL by post-processing BERT's weight.

We use two copies of pre-trained BERT weight [1], of which one is used to initialize the document understanding module. For the other copy of the 12 layer model weight, we split it for query understanding module and relevance judging module. In particular, for a l layer query understanding module, we use the first l layers of a pre-trained BERT to initialize it and use the rest of $12 - l$ layers' weight to initialize the relevance judging module. The cross-attention component in the relevance judging module borrows weights from the self-attention component by copying self-attention weights at initialization time. We note that cross-attention and the self-attention components keep separate parameters. Though both initialized with the same weight from BERT, the training procedure will diverge them for their respective jobs.

Such an initialization scheme attempts to leverage pre-trained BERT weights by replicating and splitting it into EARL's three modules. However, it has to be noted that with such an initialization method, the resulting EARL is not a real pre-trained model where the parameters are learned end-to-end. In the following sections, we will show that such initialization yields good results but do have problems.

We recognize that both query understanding and document understanding module attempt to represent natural language and utilize similar encoder block architecture. As a result, we propose to tie these two understanding modules during training and update them together to reduce the number of parameters and improve robustness. In particular, we will experiment with variants of EARL that share weight between query understanding module and the first l layers of document understanding layer.

4 Experiments Methodology

This section discusses the experimental methodologies used in this work, including datasets, baselines and experimental methods, and implementation details.

4.1 Datasets

We evaluate our model with two retrieval collections, each with two evaluation query sets.

The MS MARCO passage ranking collection (MS MARCO) (Nguyen et al., 2016) is a question-to-passage retrieval collection with 8.8M passages. The passages are around 50-60 words long. It provides a training query set that contains approximately 0.5M pairs of queries and relevant passages. On average, each query has one relevant

[1]This work uses the official bert-base-uncased weight.

passage. It also provides two evaluation query sets with distinct characteristics; this paper studied both.

- MS MARCO Dev Queries: this evaluation set contains 6980 queries. Most of the queries have only one document judged as relevant; the relevance labels are binary. Following (Nguyen et al., 2016), we used MRR@10 to evaluate the ranking accuracy on this query set.

- MS MARCO TREC2019 DL Queries: this evaluation set is the official evaluation query set used in the TREC 2019 Deep Learning Track (Craswell et al., 2019), a shared passage retrieval task. It contains 43 queries that have multiple relevant documents manually judged by NIST assessors. The TREC2019 DL Queries allow us to understand the distilled models' behavior on queries with multiple, graded relevance judgments. Follwing (Craswell et al., 2019), we used MRR, NDCG@10, and MAP@1000 as the evaluation metric.

All of our baselines and experimental rankers were evaluated in a *reranking* task to re-rank documents retrieved by a faster initial ranker. Following (Nogueira and Cho, 2019b), the rankers were tested to re-rank the top 1000 documents of the MS MARCO official BM25 retrieval results.

ClueWeb09-B is a standard document retrieval collection with 50M web pages crawled in 2009. Evaluation queries come from the TREC 2009-2012 Web Track. We used two variants of the queries:

- Title Query: this evaluation set contains 200 short, keyword-style queries that are commonly seen in standard web search engines.

- Description Query: this evaluation set contains 200 queries that are in the form of natural language statements or questions.

The experimental setup on ClueWeb09-B followed Dai and Callan (2019b). All baselines and experimental rankers were used to re-rank the initial ranking provide by Dai and Callan (2019b), which contains 100 candidate documents for each query. For handling long documents, all ranker follows the Dai and Callan (2019b)'s `BERT-FirstP` framework, which uses the title and the first passage to represent the document.

4.2 Baselines and Experimental Methods

We use two BERT-based neural rankers as baselines and compares them with different model variants of `EARL`.

The first baseline is the `BERT` ranker. It is a state-of-the-art ranker fine-tuned from BERT, which process concatenated query document pair (Nogueira and Cho, 2019a; Dai and Callan, 2019b). Its attention provides rich *interaction* signals but at the cost of computation. We use this baseline to demonstrate that the task dis-entanglement in `EARL` does not hurt the state-of-the-art ranker's performance.

The second baseline is the `BERT-Siamese` ranker, a *representation-based* ranker that encodes documents and queries each into a fixed-size embedding using BERT, with dot product as the similarity measure for ranking. In other words, it is a neural ranker that is efficient, but loses token-level information for matching. We use it to show that the existence of token-level matching captured by cross-attentive operations in `EARL`'s relevance judging module is critical for matching performance. (Reimers and Gurevych, 2019)

The two baseline models inherit weights from the 12-layer BERT base model (Devlin et al., 2019). Following (Nogueira and Cho, 2019a), we fine-tune the `BERT` ranker over ranking loss. For the `BERT-Siamese` ranker, we follow (Reimers and Gurevych, 2019) and train ranker over hinge loss. We refer readers to those two papers for details.

The experimental methods include four `EARL` variants using different pooling and weight tieing. We studied two pooling functions for `EARL`'s relevance judging model (Eq.12): ClsPool and AvePool. ClsPool takes the [CLS] token's representation to produce the final ranking score. AvePool takes the average of the token representations in the last layer. We also studied the use of weighting tying. The default `EARL` model uses separate weights for the query understanding module and the document understanding module. We test a tied variant that shares the query understanding modules weight with the corresponding layers' weight in the document understanding module. Combing the two pooling techniques and the use of weighting tieing gave us four `EARL` variants.

All `EARL` variants have document understanding module of 12 transformer layers so that it can be initialized using BERT. By default, the query understating module uses 10 transformer layers, and the relevance judging module uses 2 layers of the judger blocks. Section 5.3 provides a study on the effects of the number of judger block layers.

4.3 Implementation Details

We implemented baselines and EARL models using Pytorch (?) based on the huggingface implementation of Transformers (Wolf et al., 2019). For evaluation over the MS MARCO passage ranking dataset, we trained EARL over a subset of Marco's training set. Unless specified otherwise, the model were trained 2M training examples. We use stochastic gradient descent to train the model with a batch size of 128. We use AdamW optimizer with a learning rate of 3e-5, a warm-up of 1000 steps and a linear learning rate scheduler. As the number of training pairs in ClueWeb09-B is very limited, following (Dai and Callan, 2019b), we run a domain adaptation experiment on ClueWeb09-B: we take trained model on MS MARCO, and continue training over ClueWeb09-B's training data using 5-fold cross-validation. We use a batch size of 32 and a learning rate of 5e-6.

5 Experiment Results

Four experiments studied EARL's ranking effectiveness, its ranking speed, the effects of varying the number of judger blocks used in EARL, and the impacts of weighting initialization in EARL.

5.1 Ranking Effectiveness

Table 2 reports the ranking accuracy of EARL and two baseline BERT-based rankers.

Comparing EARL with the BERT ranker, we saw that EARL's accuracy was very close to BERT ranker in various datasets. On the MS MARCO's Dev query set, EARL was able to rerank queries with single relevance and achieves comparable performance as the BERT ranker. On MS MARCO's TREC2019 DL query set, EARL trained over MARCO can generalize to rerank candidates for queries with multiple judgments and achieve close or better performance compared to BERT ranker. The experiment on ClueWeb09-B further confirms the effectiveness of EARL. It also shows that the decoupled design of EARL does not limit its ability for domain transfer. The three modules can be jointly transferred and have close to transferred BERT ranker performance.

Comparing EARL with BERT-Siamese ranker, we found that EARL's accuracy is significantly higher regardless of model variants. This result proves our hypothesis that global document/query embeddings, although efficient, are not sufficient for accurate ranking. The attention in BERT ranker is critical for modeling interactions between query tokens and document tokens, and it is necessary to retain *token-level* representations as in EARL.

Among the four variants, we observe that weight tying can help improve robustness in model training. Average pooling, on the other hand, improve some metrics but also leads to drops in others.

In summary, this collection of results demonstrates that the computational expensive full attention over the concatenated query-document span can be substituted by EARL's light decoupled query/document attention. By storing the token-level representations instead of global query/document embeddings, EARL retains token-level matching information, outperforming the previous representation-based ranking paradigm. The EARL framework can achieve competitive reranking performance as the state-of-the-art ranking models.

5.2 Ranking Speed

The second experiment measures EARL's real-time processing speed and compared it with measurement for the BERT ranker. We record average time for ranking one query with 1000 candidate documents on both CPU and GPU. GPU test was run on a single RTX 2080 TI, and CPU in a SLURM task environment with 8 Xeon Silver 4110 logical cores. To show EARL's superiority in compute complexity, we measured ranking speed with document of length 128, 256, 512. Meanwhile, we fixed query length at 16. Recall that we proposed two reuse strategies for EARL in Section 3.3 with different time vs. time trade-offs. Table 3b(a) and (b) shows the speed tests for the two reuse strategies, correspondingly.

We observe a substantial speedup in EARL compared to BERT ranker, and the gain is consistent across CPUs and GPUs. The original BERT ranker took hundreds of seconds – several minutes – to generate results for one query on a CPU machine, which is impractical for real-time use. EARL substantially reduces the ranking

Table 2: Effectiveness of EARL models and baseline rankers on the MS MARCO Passage Ranking dataset and ClueWeb09-B dataset.

Models	MS MARCO Passage Ranking				Clueweb09-B	
	Dev Queries	TREC2019 DL Queries			Title Queries	Description Queries
	MRR @10	MRR	NDCG @10	MAP @1000	NDCG @20	NDCG @20
BERT ranker	0.3527	0.9349	0.7032	0.4836	0.3294	0.3597
BERT-Siamese ranker	0.3082	0.8418	0.5942	0.3069	–	–
EARL ClsPool	0.3442	0.9244	0.6721	0.4650	0.3316	0.3571
EARL AvePool	0.3415	0.9264	0.7041	0.4729	0.3317	0.3560
EARL ClsPool Tied	0.3456	0.9283	0.7026	0.4777	0.3323	0.3525
EARL AvePool Tied	0.3468	0.9671	0.7175	0.4748	0.3355	0.3569

Table 3: Average time in seconds to evaluate one query with 1,000 candidate documents.

(a) EARL with the Document Representation Reuse Strategy (S1)

	CPU			GPU		
Document length d	EARL w/ S1	BERT ranker	Speedup	EARL w/ S1	BERT ranker	Speedup
128	8.11	161.51	19.9x	0.12	2.70	22.01x
256	12.39	349.70	28.22x	0.19	5.78	30.61x
512	19.99	698.01	34.91x	0.32	13.05	40.20x

(b) EARL with the Transformed Document Representation Reuse Strategy (S2)

	CPU			GPU		
Document length d	EARL w/ S2	BERT ranker	Speedup	EARL w/ S2	BERT ranker	Speedup
128	4.52	161.51	35.7x	0.05	2.70	47.94x
256	5.18	349.70	67.55x	0.07	5.78	84.71x
512	5.63	698.00	123.89x	0.10	13.05	124.08x

time. Using the reuse strategy S1, EARL was $20x$ faster than the BERT ranker on shorter documents ($d = 128$). The difference is more profound on longer documents. As the length of the document increases, a larger portion of compute in BERT ranker is devoted to performing self-attention over the document sequence. EARL pre-computes document representations and avoids document-side self attention, rending up to $40x$ speedup on longer documents ($d = 512$).

Comparing the two document representation reuse strategy, we observed that strategy S2 – the transformed document reuse strategy – further enlarges the gain in speed, leading to up to $120x$ speedup. Recall that in EARL's relevance judging module, the document-related operation contains two parts: document transformation (n^2d time complexity where n is the model hidden dimension), and document-query cross attention(nqd time complexity). In practice, n is often much larger than q, e.g., our experiment used $n = 768$ while $q = 16$. S2 pre-compute the transformed document representations offline and avoids the expensive n^2d term at evaluation time.

As EARL requires data flow from understanding modules to relevance judging module, we expect a higher overhead of EARL than the BERT ranker. The ranking times reported in table 3b did not exclude those overheads, and we use them as a rough guideline to EARL's potential. We expect a lower level optimization in C++ will reduce the overheads and further speed up the execution.

5.3 Effects of Judger Block Layers

The third experiment studies the effect of varying number of judger blocks in the relevance judging module. As described in section 3, the judger block models the interaction between query and document. Within each block, a query to document attention captures query-document interactions, and a query side self-attention collects up the matching signals. Stacking up a total of

Table 4: Ranking accuracy of EARL with varying number of judger blocks. The models were tested on the MS MARCO dataset with the Dev Queries.

Model	Dev Queries MRR@10
EARL ClsPool	
1 block	0.33015
2 blocks	0.34422
3 blocks	0.34255
4 blocks	0.32732
EARL ClsPool Tied	
1 block	0.33339
2 blocks	0.34561
3 blocks	0.34226
4 blocks	0.33071

N blocks means performing such operation for N times. In this section, we investigate how the number of iterations will affect the model's behaviors.

We take two variants of EARL, EARL ClsPool and EARL ClsPool Tied, and train variations with 1, 2, 3 and 4 judger blocks respectively and evaluate performance over the MS MARCO Dev queries. The results are summarized in table 4. We see for both model variants, one judger block is less effective compared to two, suggesting that a single round of interactions is not sufficient for matching. On the other hand, model performance saturates at two judger blocks, and further increasing the number to three does not improve performance. Increasing the number of blocks to four hurts the model's effectiveness. The decrease could be caused by our current way of initializing EARL. As we are not able to pre-train EARL from scratch due to hardware constraints, the judger blocks are initialized by borrowing parameters from a pre-trained BERT. These borrowed weights may not well align with the other parts of the EARL and can interrupt the hidden states, making the model starting point worse and the optimization problem harder.

5.4 Impacts of Weight Initialization

Recall that in our proposed weight initialization scheme (Section 3.4), which utilizes BERT weight, for relevance judging module, we copy self-attention weight to cross attention, to avoid random initialization and to ease optimization. In this section, we take EARL models initialized with our proposed weight initialization scheme and compared them with ones that have cross attention weights initialized randomly. We ablate two variants of EARL, EARL ClsPool and EARL ClsPool Tied, with randomly initialized judger block cross attention weight while holding other training setting the same as described in 4.3. We evaluate the ablated models over the MS Marco Dev Queries and compare the result with their unablated counterparts in table 5. Here, we see a drop in performance when cross attention weights are randomly initialized, indicating the importance of our proposed initialization method. With such a deep transformer model, a set of uninitialized weights can largely increase optimization difficulty.

Table 5: Ranking Accuracy of EARL when using / not using attention weights copied from BERT to initialize cross attention in the relevance judging module. The models were tested on the MS MARCO dataset with the Dev Queries.

Model	Dev Queries MRR@10
EARL ClsPool	
w/ copy	0.3442
w/o copy	0.2723
EARL ClsPool Tied	
w/ copy	0.3456
w/o copy	0.2414

6 Conclusion

In this paper, we propose EARL, a framework that enables fast ranking with transformers. EARL decouples document understanding, query understanding, and relevance judging. As a result, EARL can build document representations beforehand. We further propose two representation reuse strategies that can take advantage of EARL as well as an initialization technique that helps EARL to leverage pre-trained weights from BERT. Our experiments demonstrate that EARL can achieve comparable performance with state-of-the-art BERT-based rankers but has a lower complexity and, in practice, runs tens to over a hundred times faster.

References

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a Siamese time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744.

William R. Caid, Susan T. Dumais, and Stephen I. Gallant. 1995. Learned vector-space models for document retrieval. *Information Processing & Management*, 31(3):419–429.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2019. Overview of the trec 2019 deep learning track. In *TREC (to appear)*.

Zhuyun Dai and Jamie Callan. 2019a. Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv preprint arXiv:1910.10687*.

Zhuyun Dai and Jamie Callan. 2019b. Deeper text understanding for ir with contextual neural language modeling. In *The 42nd International ACM SIGIR Conference on Research & Development in Information Retrieval*.

Zhuyun Dai, Chenyan Xiong, James P. Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM '18*.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27*, pages 2042–2050.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management*, pages 2333–2338.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Rodrigo Nogueira and Kyunghyun Cho. 2019a. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Rodrigo Nogueira and Kyunghyun Cho. 2019b. Passage re-ranking with BERT. *arXiv:1901.04085*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP/IJCNLP*.

Gerard Salton and Michael McGill. 1984. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International World Wide Web Conference*, pages 373–374.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zhijing Wu, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2019. Investigating passage-level relevance and its role in document-level relevance judgment. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Chenyan Xiong, Zhuyun Dai, James P. Callan, Zhiyuan Liu, and Russell Power. 2017a. End-to-end neural ad-hoc ranking with kernel pooling. *ArXiv*, abs/1706.06613.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017b. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 55–64.