Fast Forward Indexes for Efficient Document Ranking

Jurek Leonhardt  
L3S Research Center  
Hannover, Germany  
leonhardt@L3S.de

Koustav Rudra  
L3S Research Center  
Hannover, Germany  
rudra@L3S.de

Megha Khosla  
L3S Research Center  
Hannover, Germany  
khosla@L3S.de

Abhijit Anand  
L3S Research Center  
Hannover, Germany  
anand@L3S.de

Avishek Anand  
L3S Research Center  
Hannover, Germany  
anand@L3S.de

ABSTRACT
Neural approaches, specifically transformer models, for ranking documents have delivered impressive gains in ranking performance. However, query processing using such over-parameterized models is both resource and time intensive. Consequently, to keep query processing costs manageable, trade-offs are made to reduce the number of documents to be re-ranked or consider leaner models with fewer parameters.

In this paper, we propose the fast-forward index – a simple vector forward index that facilitates ranking documents using interpolation-based ranking models. Fast-forward indexes pre-compute the dense transformer-based vector representations of documents and passages for fast CPU-based semantic similarity computation during query processing. We propose theoretically grounded index pruning and early stopping techniques to improve the query-processing throughput using fast-forward indexes. We conduct extensive large-scale experiments over the TREC-DL datasets and show up to 75% improvement in query-processing performance over hybrid indexes using only CPUs. Along with the efficiency benefits, we show that fast-forward indexes can deliver superior ranking performance due to the complementary benefits of interpolation between lexical and semantic similarities.

KEYWORDS
retrieval, dense, sparse, hybrid, ranking, interpolation

1 INTRODUCTION
Ad-hoc document ranking is a central task in web search and information retrieval, where the objective is to rank documents relevant to a user-specified keyword query. The standard approach for ranking documents follows the retrieve-and-re-rank framework, where the retrieval phase focuses on a fast high-recall candidate selection stage followed by the more computationally intensive re-ranking phase. Recent approaches for ranking documents have focused heavily on neural transformer-based models for both retrieval [6, 18, 23] and re-ranking [1, 4, 12, 14, 21, 28].

The predominant strategy for the retrieval step is based on term-based or lexical matching. Towards efficient retrieval, inverted indexes have been employed as workhorses in traditional information retrieval. Inverted indexes, referred to as sparse indexes, exploit sparsity in the term space for pruning out a large number of irrelevant documents. Alternately, dense indexes have been recently proposed, which independently vectorize query and document using dual-encoder transformer models. We note that, irrespective of the retrieval methodology, the time taken for re-ranking is often a bottleneck in terms of query processing throughput. This high processing cost is primarily due to the use of over-parameterized cross-attention models for scoring each retrieved document. Consequently, to keep ranking costs manageable, either smaller number of documents are considered in the re-ranking phase or leaner models with lesser number of parameters are used. Both these design decisions result in reduction of ranking performance.

In this paper, we aim to drastically reduce query processing costs by eliminating the need to run inference through transfer-based ranking models during the re-ranking phase. Our idea is simple and builds upon the improved ranking performance of interpolation-based ranking models [1]. Interpolation-based rankers score documents based on a linear combination of the semantic similarity scores (from the re-ranking phase) and lexical matching (from the retrieval phase) with the query. We make two crucial novel observations that allow us to greatly improve query processing time during re-ranking.

Fast-Forward Indexes. We find that dual-encoder models can be used to compute semantic similarity in interpolation-based ranking, delivering superior ranking performance. In fact, they perform much more favourably in comparison to cross-attention models. Exploiting this observation, we propose a simple vector forward index of documents referred to as the FAST-FORWARD index, which contains a list of vectors for each document corresponding to the encodings of its constituent passages. Query processing using a FAST-FORWARD index entails computing the semantic similarity of the retrieved documents, typically by a sparse index, by a sequence of look-ups and dot product computations. The distinct advantage of using a forward index is that we replace the computationally expensive inference (typically computed on GPUs) by efficient look-up operations (computed on CPUs).

Query Processing Techniques. We further propose two important techniques to improve efficiency in query processing over FAST-FORWARD indexes – sequential coalescing and early stopping. In sequential coalescing, we substantially reduce the number of vectors per document by combining the representations corresponding to adjacent yet similar passages. This not only improves the query processing efficiency but also reduces the memory footprint of the index. Early stopping exploits the natural ordering of the sparse scores to avoid unnecessary index accesses by maintaining a top-\(k\) score estimate for the dense scores. Specifically, we access the retrieved documents in the order of the sparse scores and compute the interpolated scores of the documents by forward index look-ups.
Experimental Evaluation. We perform extensive experimental evaluation to show the performance benefits of fast-forward indexes and our query-processing techniques. We find that interpolation using dual-encoder models consistently yield better performance than standard re-ranking using the same models. Further, increasing the sparse retrieval depth prior to interpolation consistently improves the final ranking. Finally, we show how optimized FAST-FORWARD indexes accelerate the re-ranking, decreasing the query processing time by up to 75% compared to hybrid retrieval, while maintaining comparable performance.

Summary of Contributions. To sum up, we make the following contributions:

- We propose FAST-FORWARD indexes as an efficient re-ranking approach for ad-hoc document ranking tasks.
- We propose novel query-processing algorithms – sequential coalescing and early stopping – that further improve the overall query processing throughput.
- We perform extensive experimental evaluation to establish the strong efficiency gains due to our forward indexes and query-processing techniques.

2 RELATED WORK

In the following sections we discuss the use of contextual models in both first-stage retrieval and the re-ranking phase.

2.1 Contextual Models in First-Stage Retrieval

2.1.1 Sparse Retrieval. Classical ranking approaches, such as BM25 or the Query Likelihood Model [20], rely on the inverted index for efficient first-stage retrieval that stores term-level statistics like term-frequency, inverse document frequency and positional information. We refer to this style of retrieval as sparse retrieval, since it assumes sparse document representations. Instead of storing term frequencies, recently Dai and Callan [6, 7] proposed DEEP-CT, which stores contextualized scores for terms in the inverted index for both passage and document retrieval. Similar to DEEP-CT, SPLADE [8] aims to enrich sparse document representation using a trained, contextual transformer model using sparsity regularization on the term weights. In this work, we focus on the vanilla inverted index with standard term statistics for first-stage retrieval.

2.1.2 Dense Retrieval. Although DEEP-CT learns better term weights, the final retrieval is still restricted to the term-based matching, thereby failing to retrieve semantically related documents that do not contain the query terms. An alternate design strategy is to use dual-encoders to learn dense vector representations for passages or documents using contextual models [10, 17, 18]. The dense vectors are then indexed in an offline phase [16], where retrieval is akin to performing an approximate nearest neighbor search (ANN) given a vectorized query. Consequently, there has been a large number of follow-up works that improve the performance of dual-encoder models by improved pre-training [2], better optimization techniques [11] and improved negative sampling [33, 38]. In this work, we use dual-encoders for computing semantic similarity between queries and passages. Some approaches have also proposed architectural modifications to the dual-encoder models by having lightweight aggregations between the query and passage embeddings [3, 12, 15, 24, 25, 38, 39], showing promising performance over standard term-based retrieval strategies.

2.2 Models for Semantic Similarity

While lexical matching models are typically employed in the first-stage retrieval and are known to achieve high recall, the ability to accurately determine semantic similarity is essential in the subsequent more involved and computationally expensive re-ranking stage to improve the vocabulary mismatch problem [5, 7, 27, 29, 32]. Computing the semantic similarity of a document given a query has been heavily researched in IR using smoothing methods [19], topic models [37], embeddings [31], personalized models [26] and other techniques. In these classical approaches, ranking is performed by interpolating the semantic similarity scores with the lexical matching scores from first-stage retrieval. Recent approaches have been dominated by over-parameterized contextual models that have been used predominantly in re-ranking [1, 4, 12, 14, 21, 28]. Unlike dual encoder models used in dense retrieval, most of the above ranking models take as input a concatenation of the query and document. This combined input results in higher query processing times for large retrieval depths since each document has to be processed in conjunction with the query string. Another key limitation of using contextual models for document ranking is the maximum acceptable number of input tokens for transformer models. Some strategies address this length limitation by document truncation [28], and chunking into passages [4, 34]. However, performance of chunking-based strategies depends very much on the chunking properties, i.e. passage length or overlap among consecutive passages [35]. Recent proposals include a two stage approach where they first generate the query-specific summary of the document, i.e. they retrieve relevant portion of the documents and next, followed by applying re-ranking strategies over the query and summarized document [13, 22]. The summarized document fits well within the token limit.

2.2.1 Interpolation-based Rankers. Unlike classical methods, where score interpolation is the norm, semantic similarity using contextual models is not consistently combined with the matching scores. Recently, Wang et al. [36] showed that the interpolation of BERT-based retrievers and sparse retrieval methods can boost the performance. Further, they analyzed the role of interpolation in BERT-based dense retrieval strategies (ANCE, RepBERT) and found that dense retrieval alone is not enough, but interpolation with BM25 score is necessary for proper working of dense retrieval strategy.

3 INTERPOLATION-BASED RE-RANKING

In this section we formally introduce the problem of re-ranking. We further compare traditional and interpolation-based re-ranking.

3.1 Problem Statement

The retrieval of documents or passages given a query typically happens in two stages: In the first stage, a term-frequency-based (sparse) retrieval method retrieves a set of documents from a large corpus. In the second stage, another model, which is usually much more computationally expensive, re-ranks the retrieved documents again. The re-ranking step is deemed very important for tasks that require high performance for small retrieval depths. For example, in a search
Table 1: Performance comparison (nDCG@10) between re-ranking and interpolation at retrieval depth $k = 1000$. Interpolation-based re-ranking, as described in Section 3, like standard re-ranking, is known to improve performance when applied to the results of a first-stage (sparse) retrieval step. However, many re-rankers are notoriously inefficient compared to the first-stage retrieval step, i.e. the computation of $\phi_D$ (cf. Eq. 2) is very expensive, where query-document based cross-attention models [4, 34] are more expensive than dual-encoder-based query-document ranking strategies [2, 25]. In this section we propose several means of implementing interpolation-based re-ranking more efficiently.

### 4.1 Hybrid Retrieval

Hybrid retrieval is similar to standard interpolation-based re-ranking (cf. Section 3), however, the key difference is that the dense scores $\phi_S(q, d)$ are not computed for all query-document pairs. Instead, this approach operates under the assumption that $\phi_D$ is a dense retrieval model, which retrieves documents $d_i$ and their corresponding scores $\phi(q, d_i)$ using (approximated) nearest neighbor search given a query $q$. Hybrid retrieval aims to combine the retrieved sets of a sparse and a dense retriever efficiently.

For a query $q$, we start by retrieving (in parallel) two sets of documents, $K^q_S$ and $K^q_D$, using the sparse and dense retrievers respectively. We denote the scoring functions by $\phi_S$ and $\phi_D$. Note that the two retrieved sets are usually not equal, i.e. most of the documents only have a single score. One strategy proposed in [25] ranks all documents in $K^q_S \cup K^q_D$ approximating missing scores. In our experiments, however, we found that only considering documents from $K^q_S$ for the final ranking and discarding the rest works well. The final score of a query-document pair is thus computed as follows:

$$
\phi(q, d) = \alpha \cdot \phi_S(q, d) + (1 - \alpha) \cdot \phi_D(q, d)
$$

(3)

The re-ranking step in hybrid retrieval is essentially a sorting operation over the interpolated scores and takes negligible time (less than 5ms) in comparison to standard re-ranking.

### 4.2 Fast Forward Indexes

The hybrid approach described in Section 4.1 has two distinct disadvantages. Firstly, in order to retrieve $K^q_S$, an (approximate) nearest neighbor search has to be performed, which is time consuming. Secondly, some of the query-document pairs are missed, leading to an incomplete interpolation.

In this section we propose FAST FORWARD indexes as an efficient way of computing dense scores for known documents that alleviates the aforementioned issues. Specifically, FAST FORWARD indexes build upon two-tower dense retrieval models that compute the score of a query-document pair as a dot product

$$
\phi_D(q, d) = \zeta(q) \cdot \eta(d),
$$

(4)
where $\zeta$ and $\eta$ are the query and document encoders, respectively. Examples of such models are ANCE [38] and TCT-Cot.BERT [25]. Since the query and document representations are independent for two-tower models, we can pre-compute the document representations $\eta(d)$ for each document $d$ in the corpus. These document representations are then stored in an efficient hash map, allowing for look-ups in constant time. After the index is created, the score of a query-document pair can be computed as

$$\phi_D^F(q, d) = \zeta(q) \cdot \eta^F(d), \quad (5)$$

where the superscript $F$ indicates the look-up of a pre-computed document representation in the FAST-FORWARD index. At retrieval time, only $\zeta(q)$ needs to be computed once for each query. Note that, for maxP indexes (cf. Section 3.1), the score is computed as follows:

$$\phi_D^F(q, d) = \max_{p \in d} (\zeta(q) \cdot \eta^F(p)) \quad (6)$$

### 4.3 Index Compression via Sequential Coalescing

A major disadvantage of dense indexes and dense retrieval in general is the size of the final index. This is caused by two factors: Firstly, in contrast to sparse indexes, the high-dimensional dense representations can not be stored as efficiently as sparse vectors. Secondly, the dense encoders are typically transformer-based, imposing a (soft) limit on their input lengths due to their quadratic time complexity with respect to the inputs. Since documents usually exceed this limit, they are split into passages prior to indexing, where each passage is encoded separately (maxP indexes).

As an increase in the index size has a negative effect on retrieval latency, both for classical nearest neighbor search and FAST-FORWARD indexing as used by our approach, we exploit a sequential coalescing approach as a way of dynamically combining the representations of consecutive passages within a single document in maxP indexes. The idea is to reduce the number of passage representations in the index for a single document. To that end, we employ the cosine distance function and a threshold parameter $\delta$ that controls the degree of coalescing. Within a single document, we iterate over its passage vectors in their original order and maintain a set $\mathcal{A}$, which contains the representations of the already processed passages, and continuously compute $\overline{\mathcal{A}}$ as the average of all vectors in $\mathcal{A}$. For each new passage vector $v$, we compute its cosine distance to $\overline{\mathcal{A}}$. If it exceeds the distance threshold $\delta$, the current passages in $\mathcal{A}$ are combined as their average representation $\overline{\mathcal{A}}$. Afterwards, the combined passages are removed from $\mathcal{A}$ and $\overline{\mathcal{A}}$ is recomputed. This approach is illustrated in Algorithm 1. Figure 1a shows an example index after coalescing.

### 4.4 Accelerating Interpolation by Early Stopping

As described in Section 3, by interpolating the scores of sparse and dense retrieval models we perform implicit re-ranking, where the dense scores are pre-computed and can be looked up in a FAST-FORWARD index at retrieval time. Further, increasing the sparse retrieval depth $k_S$, such that $k_S > k$, where $k$ is the final number of documents, improves the performance. A drawback of this is that an increase in the number of retrieved documents also results in an increase in the number of necessary look-ups in the index.

In this section we propose an extension to FAST-FORWARD indexes that allows for early stopping, i.e. avoiding a number of unnecessary look-ups, for cases where $k_S > k$ by approximating the
The approach is illustrated in Algorithm 2 and Figure 1b. Since the actual top-\( k \) maximum of the dense scores. Then the returned list of scores are the possible best score \( s_{\text{best}} \) is computed using the sparse score found next in the decreasing sequence and the maximum of all dense scores, \( s_{\min} \) (cf. line 7). If \( s_{\text{best}} \) is less than the minimum of the scores in \( Q \), then \( Q \) already contains the top-\( k \) scores. To see this, note that the first component of \( s_{\text{best}} \) is the largest among all unseen sparse scores (as the list is sorted) and \( s_{\min} \) is maximum of the dense scores by our assumption.

Next, we show that a good approximation of the top-\( k \) scores can be achieved by using the sample maximum. To prove our claim, we use the following Dvoretzky–Kiefer–Wolfowitz (DKW) [30] inequality.

**Lemma 4.2.** Let \( x_1, x_2, ..., x_n \) be \( n \) real-valued independent and identically distributed random variables with the cumulative distribution function \( F(\cdot) \). Let \( F_n(\cdot) \) denote the empirical cumulative distributive function, i.e.

\[
F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{x_i \leq x}, \quad x \in \mathbb{R}. 
\]

Then, according to the DKW inequality, the following one-sided estimate holds:

\[
\Pr\left( \sup_{x \in \mathbb{R}} (F_n(x) - F(x)) > \epsilon \right) \leq e^{-2n\epsilon^2} \forall \epsilon \geq \sqrt{\frac{1}{2n} \ln 2}. \tag{10}
\]

In the following we show that, if \( s_{\min} \) is chosen as the maximum of a large random sample drawn from the set of dense scores, then the probability that any given dense score, chosen independently and uniformly at random from the dense scores, is greater than \( s_{\min} \) is exponentially small in the sample size.

**Theorem 4.3.** Let \( x_1, x_2, ..., x_n \) be \( n \) real-valued independent and identically distributed random variables drawn from the distribution of the dense scores with the cumulative distribution function \( F(\cdot) \). Let \( z = \max(x_1, x_2, ..., x_n) \). Then, for every \( \epsilon > \frac{1}{\sqrt{2n}} \ln 2 \), we obtain

\[
\Pr(F(z) < 1 - \epsilon) \leq e^{-2n\epsilon^2}. \tag{11}
\]

**Proof.** First, note that the sparse scores, \( \phi_S(q, d_i) \), are already sorted in decreasing order for a given query. By construction, the priority queue \( Q \) always contains the top-\( k \) scores corresponding to the list parsed so far. Let, after parsing \( t \) scores, \( Q \) be full. Now the possible best score \( s_{\text{best}} \) is computed using the sparse score found next in the decreasing sequence and the maximum of all dense scores, \( s_{\min} \) (cf. line 7). If \( s_{\text{best}} \) is less than the minimum of the scores in \( Q \), then \( Q \) already contains the top-\( k \) scores. To see this, note that the first component of \( s_{\text{best}} \) is the largest among all unseen sparse scores (as the list is sorted) and \( s_{\min} \) is maximum of the dense scores by our assumption.

**4.4.1 Theoretical Analysis.** We first show that the early stopping criteria, when using the true maximum of the dense scores, is sufficient to obtain the top-\( k \) scores.

**Theorem 4.1.** Let \( s_{\min} \) as used in Algorithm 2 be the true maximum of the dense scores. Then the returned list of scores are the actual top-\( k \) scores.
Besides the above result, we exploit the following empirical facts to further simplify the early stopping criteria: Firstly, the sparse scores are sorted and show a higher variation (empirically) than the dense scores. Second, we observe that the dense scores are positively correlated with the sparse scores. Therefore, rather than using a random sample to obtain \( s_p \), we go through the list of scores (sorted by sparse scores) and update \( s_p \) based on the samples seen so far.

5 EVALUATION SETUP

We consider the following baselines:

- **Lexical or sparse retrievers** rely on term-based matching between queries and documents. We consider BM25, which uses standard term based retrieval signals, and DEEP-CT [6], which is similar to BM25, but the term weights are learned in a contextualized fashion.

- **Semantic or dense retrievers** map queries and documents to a common embedding space and retrieve documents that are semantically similar to the queries in that space. We consider TCT-ColBERT [25] and ANCE [38]. Both approaches are based on BERT encoders. Large documents are split into passages before indexing (maxP).

- **Hybrid retrievers** interpolate sparse and dense retriever scores. We consider CLEAR [11], a retrieval model that complements lexical models with semantic matching. Additionally, we consider the hybrid strategy described in Section 4.1 as a baseline. These dense retrievers use exact maximum inner product search (MIPS), as opposed to approximate nearest neighbor search.

- **Contextual dense re-rankers** operate on the documents retrieved by a sparse retriever, such as BM25. Each query-document pair is input into the re-ranker, which outputs a corresponding score. In this paper, we use a BERT-CLS re-ranker, where output corresponding to the classification token is used as the score. Note that re-ranking is performed using the full documents (i.e. documents are not split into passages). If an input exceeds 512 tokens, it is truncated.

**Datasets.** We conduct experiments on three datasets from the TREC Deep Learning track, Doc’19, Doc’20 and PASSAGE’19, to evaluate the effectiveness and efficiency of retrieval and re-ranking strategies. Each testset has a total of 200 queries. The documents and passages are retrieved from the MS MARCO collection. We use the PYSERINI toolkit [24] for our retrieval experiments.

**Hyperparameters.** We use the MS MARCO development set to determine the best value of \( \alpha \) for interpolation. We set \( \alpha = 0.2 \) for TCT-ColBERT, \( \alpha = 0.5 \) for ANCE and \( \alpha = 0.7 \) for BERT-CLS throughout all of our experiments. We report latency per query as the sum of scoring, interpolation and sorting cost. Tokenization cost is omitted. Where applicable, dense models use a batch size of 256.

6 EXPERIMENTAL RESULTS

In this section we perform large-scale experiments to show the effectiveness and efficiency of the proposed FAST-FORWARD indexes.

**RQ1.** How does interpolation-based re-ranking with dual-encoders compare to other methods? In Table 2, we report the performance of standard dense re-rankers, hybrid retrieval and interpolation. We observe several interesting patterns.

First, we observe that dense retrieval strategies perform better than sparse ones in terms of nDCG but have poor recall except on PASSAGE’19. The contextual weights learned by DEEP-CT are better than tf-idf based retrieval (BM25), but fall short of dense semantic retrieval strategies (TCT-ColBERT and ANCE). However, the overlap among retrieved documents is pretty low, reflecting that dense retrieval cannot match query and document terms well. Besides, the latency of dense retrievers is significantly higher, as they rely on maximum inner product search.

Second, dual-encoder based re-rankers (TCT-ColBERT and ANCE) perform better than contextual re-rankers (BERT-CLS). In this setup, we first retrieve \( k_S = 1000 \) documents using a sparse retriever and re-rank them. This approach benefits from a high recall from the first stage and promotes the relevant documents to the top of the list through the dense semantic re-ranker. However, re-ranking is typically time-consuming and requires GPU acceleration. The improvement of TCT-ColBERT and ANCE over BERT-CLS also suggests that dual-encoder-based re-ranking strategies are better than cross-interaction-based methods. However, the difference could also be attributed to the fact that BERT-CLS re-ranks the full documents, while the other follow the maxP approach (cf. Section 3.1).

Finally, interpolation-based re-ranking, which combines the benefits of sparse and dense scores, significantly outperforms the BERT-CLS re-ranker and dense retrievers. Recall that dense re-rankers operate solely based on the dense scores and discard the sparse BM25 scores of the query-document pairs. The superiority of interpolation-based methods is also supported by evidence from recent studies [2, 3, 10, 11].

**RQ2.** Do FAST-FORWARD indexes allow for efficient interpolation at higher retrieval depths? Tables 3 and 4 show results of re-ranking, hybrid retrieval and interpolation on document and passage datasets, respectively. Each metric is computed for two sparse retrieval depths, \( k_S = 1000 \) and \( k_S = 5000 \). We summarize our findings in the following.

We observe that taking the sparse component into account (as is done in the interpolation and hybrid methods) in the score computation causes interpolation performance to improve with retrieval depth. Specifically, some queries receive a considerable recall boost, capturing more relevant documents with large retrieval depths. Interpolation based on FAST-FORWARD indexes achieve substantially lower latency compared to all other methods. Pre-computing the document representations allows for fast look-ups during retrieval time. As only the query needs to be encoded by the dense model, both retrieval and re-ranking can be performed on the CPU while still offering considerable improvements in query processing time. Next, we will show how FAST-FORWARD indexes can be optimized to decrease the latency even further.

The hybrid retrieval strategy as described in Section 4.1 shows good performance, being close to interpolation-based methods in most cases. However, as the dense indexes require maximum inner product search for retrieval, the query processing latency (although computed on the CPU) is much higher than for interpolation using FAST-FORWARD indexes.

Finally, dense re-rankers do not profit reliably from increased sparse retrieval depth; on the contrary, the performance drops in some cases. This trend is more apparent for the document retrieval
Dense Retrieval
TCT-COBERT 0.279 0.576 0.612\(^1\)
ANCE 0.254 0.510 0.633\(^1\)

Hybrid
CLEAR - - - - - - 0.511 0.812 0.699 -

Re-Ranking
TCT-COBERT 0.370 0.697 0.685
ANCE 0.336 0.697 0.654
BERT-CLS 0.283 0.697 0.528\(^1\)

Interpolation
TCT-COBERT\(^1\) 0.406 0.697 0.696
ANCE\(^2\) 0.387 0.697 0.673
BERT-CLS\(^3\) 0.365 0.697 0.612

Table 2: Retrieval performance for depth \(k_s = 1000\). Scores for CLEAR and DEEP-CT are taken from the corresponding papers [10, 11]. Latency is reported per query for PASSAGE’19 on CPU and GPU. Superscripts indicate statistically significant improvements using two-paired tests with a sig. level of 95% [9].

| k\(_s\) = 1000 | k\(_s\) = 5000 |
|----------------|----------------|
| AP\(_k\) | R\(_k\) | nDCG\(_k\) | AP\(_k\) | R\(_k\) | nDCG\(_k\) |
| TCT-COBERT | 0.370 | 0.697 | 0.632 | 0.334 | 0.703 | 0.609\(^1\) | 0.414 | 0.809 | 0.587\(^1\) | 0.405 | 0.794 | 0.585\(^{1,3,4}\) |
| ANCE | 0.336 | 0.697 | 0.614 | 0.304 | 0.647 | 0.607 | 0.426 | 0.809 | 0.595\(^3\) | 0.422 | 0.761 | 0.604 | 1189 + 2 |
| BERT-CLS | 0.283 | 0.697 | 0.528\(^1\) | 0.159 | 0.559 | 0.289 | 0.329 | 0.809 | 0.512\(^1\) | 0.221 | 0.727 | 0.375\(^1\) | 185 + 2 |

Hybrid
TCT-COBERT 0.394 0.697 0.655
ANCE 0.379 0.697 0.633

Interpolation
TCT-COBERT\(^1\) 0.405 0.697 0.655
FAST-FORWARD optimized\(^2\) 0.379 0.697 0.630
ANCE\(^3\) 0.387 0.697 0.638
FAST-FORWARD optimized\(^4\) 0.372 0.697 0.625
BERT-CLS\(^5\) 0.365 0.697 0.585

Table 3: Document retrieval performance of baselines and interpolation methods. Latency is reported per query for \(k_s = 5000\) on CPU and GPU. The optimized FAST-FORWARD indexes are compressed to approximately 25% of their original size using sequential coalescing. HYBRID retrievers use a dense retrieval depth of \(k_D = 1000\). Superscripts indicate statistically significant improvements using two-paired tests with a sig. level of 95% [9].

datasets, where AP, recall, and nDCG fail to improve for higher values of \(k_S\). We hypothesize that dense rankers only focus on semantic matching and are sensitive to topic drift, causing them to rank irrelevant documents in the top 5000 higher.

RQ3. Can the re-ranking efficiency be improved by reducing the Fast-Forward index size using sequential coalescing? In order to evaluate this approach, we first take the pre-trained TCT-COBERT dense index of the MS MARCO corpus, apply sequential coalescing...
with varying values for \( \delta \) and evaluate each resulting compressed indexes using the Doc’19 testset. The results are illustrated in Figure 2. It is evident that, by combining the passage representations, the number of vectors in the index can be reduced by more than 80% in the most extreme case, where only a single vector per document remains. At the same time, the performance is negatively correlated with the granularity of the representations. However, the drops are relatively small. For example, for \( \delta = 0.025 \), the index size is reduced by more than half, while the nDCG decreases by roughly 0.015 (3%).

Additionally, Table 3 shows the detailed performance of coalesced Fast-Forward indexes on the document datasets. We chose the indexes corresponding to \( \delta = 0.035 \) (TCT-ColBERT) and \( \delta = 0.003 \) (ANCE), both of which are compressed to approximately 25% of their original size. This is reflected in the query processing latency, which is reduced by more than half. The overall performance drops to some extent, as expected, however, these drops are not statistically significant in all but one case. The trade-off between latency (index size) and performance can be controlled by varying the threshold \( \delta \).

\[ \delta = \alpha \]  

RQ4. Can the re-ranking efficiency be improved by limiting the number of Fast-Forward look-ups? We start by evaluating the utility of the early stopping approach described in Section 4.4 on the PASSAGE’19 dataset. Figure 3 shows the average number of lookups performed in the Fast-Forward index during interpolation w.r.t. the cut-off depth \( k \). We observe that for \( k = 100 \), early stopping already leads to a reduction of almost 20% in the number of lookups. Decreasing \( k \) further leads to a significant reduction of look-ups, resulting in improved query processing latency. As lower cut-off depths (i.e. \( k < 100 \)) are typically used in downstream tasks, such as question answering, the early stopping approach for low values of \( k \) turns out to be particularly helpful.

Table 4 shows early stopping applied to the passage dataset to retrieve the top-10 passages and compute reciprocal rank. It is evident that, even though the algorithm approximates the maximum dense score (cf. Section 4.4), the resulting rankings are identical, which means that the approximation was accurate in both cases and did not incur any performance hit. Further, the query processing time is decreased by up to a half compared to standard interpolation. Note that early stopping depends on the value of \( \alpha \), hence the latency varies between TCT-ColBERT and ANCE.

7 CONCLUSION

In this paper we propose Fast-Forward indexes, a simple yet effective and efficient look-up-based interpolation method that combines document retrieval and re-ranking. Fast-Forward indexes are based on dense dual-encoder models, exploiting the fact that document representations can be pre-processed and stored, providing efficient access in constant time. Using interpolation, we observe increased performance compared to hybrid retrieval and up to 75% improvements in query processing latency and memory footprint due to our optimization techniques, sequential coalescing and early stopping. At the same time, our method requires solely CPU computations, completely eliminating the need for expensive GPU-accelerated re-ranking.
REFERENCES

[1] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-Domain Modeling of Sentence-Level Evidence for Document Retrieval. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 3481–3487.

[2] Wei-Cheng Chang, Felix X Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2020. Pre-training tasks for embedding-based large-scale retrieval. arXiv preprint arXiv:2002.03932 (2020).

[3] Xiaoyang Chen, Kai Hui, Ben He, Xinpei Han, Le Sun, and Zheng Ye. 2021. Co-BERT: A Context-Aware BERT Retrieval Model Incorporating Local and Query-specific Context. arXiv preprint arXiv:2104.08523 (2021).

[4] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR using Contextual Neural Language Modeling. In ACM SIGIR '19, 985–988.

[5] Zhuyun Dai and Jamie Callan. 2019. An Evaluation of Weakly-Supervised DeepCT in the TREC 2019 Deep Learning Track. In TREC.

[6] Zhuyun Dai and Jamie Callan. 2020. Context-Aware Document Term Weighting for Ad-Hoc Search. In Proceedings of The Web Conference 2020. 1897–1907.

[7] Zhuyun Dai and Jamie Callan. 2020. Context-Aware Term Weighting For First Stage Passage Retrieval. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1533–1536.

[8] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (Virtual Event, Canada) (SIGIR ’21). Association for Computing Machinery, New York, NY, USA, 2288–2292. https://doi.org/10.1145/3404835.3463098

[9] F. Goharian, and Ophir Frieder. 2020. Expansion via Prediction of Importance with Contextualized Word Representations for Document Re-ranking. arXiv preprint arXiv:2004.07094 (2019).

[10] Joel Mackerzie, Zhuyun Dai, Luke Gallagher, and Jamie Callan. 2020. Efficiency Implications of Term Weighting for Passage Retrieval. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1821–1824.

[11] Rosal Massart. 1990. The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. The annals of Probability (1990), 1269–1283.

[12] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. 2016. A dual embedding space model for document ranking. arXiv preprint arXiv:1602.01137 (2016).

[13] Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, and Emine Yilmaz. 2019. Incorporating query term independence assumption for efficient retrieval and ranking using deep neural networks. arXiv preprint arXiv:1907.03691 (2019).

[14] Pratul Prakash, Julian Killinback, and Hamed Zamanli. 2021. Learning Robust Dense Retrieval Models from Incomplete Relevance Labels. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. 1728–1732.

[15] Konstantin Rudra and Avishak Anand. 2020. Distant Supervision in BERT-Based Adhoc Document Retrieval. 2197–2200.

[16] Konstantin Rudra, Zeon Trevor Fernando, and Avishak Anand. 2021. An In-depth Analysis of Passage-Level Label Transfer for Contextual Document Ranking. CoRR abs/2103.16699 (2021).

[17] Shuai Wang, Shengyao Zhan, and Guido Zuccon. 2021. BERT-based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval. In Proceedings of The 11th International Conference on the Theory of Information Retrieval. 8 pages.

[18] Xing Wei and W. Bruce Croft. 2006. LDA-Based Document Models for Ad-Hoc Retrieval. In Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (Seattle, Washington, USA) (SIGIR ’06). Association for Computing Machinery, New York, NY, USA, 178–185. https://doi.org/10.1145/1148170.1148204

[19] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor nega-tive contrastive learning for dense text retrieval. arXiv preprint arXiv:2007.00808 (2020).

[20] Jinqing Zhu, Jiaxin Yao, Lan Guan, Jun Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. arXiv preprint arXiv:2104.08053 (2021).

[21] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: An easy-to-use Python toolkit to support replicable IR research with sparse and dense representations. arXiv preprint arXiv:2102.10077 (2021).

[22] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. arXiv preprint arXiv:2010.11136 (2020).

[23] Julia Luxenburger, Shady Elbassuoni, and Gerhard Weikum. 2008. Matching Task Profiles and User Needs in Personalized Web Search. In Proceedings of the 17th ACM Conference on Information and Knowledge Management (Napa Valley, California, USA) (CIKM ’08). Association for Computing Machinery, New York, NY, USA, 689–698. https://doi.org/10.1145/1458082.1458175

[24] Sean MacAvaney, Franco Marie Nardinii, Raffaele Pergo, Nicola Tomeliotto, Nazli Goharian, and Ophir Frieder. 2020. Expansion via Prediction of Importance with Contextualization. arXiv preprint arXiv:2004.14245 (2020).

[25] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: An easy-to-use Python toolkit to support replicable IR research with sparse and dense representations. arXiv preprint arXiv:2102.10077 (2021).