# Document Expansions and Learned Sparse Lexical Representations for MS MARCO V1 and V2

Xueguang Ma*
Ronak Pradeep*
Rodrigo Nogueira
Jimmy Lin

David R. Cheriton School of Computer Science,
University of Waterloo
Waterloo, Ontario, Canada

## ABSTRACT

With doc2query, we train a neural sequence-to-sequence model that, given an input span of text, predicts a natural language query that the text might answer. These predictions can be viewed as document expansions that feed standard bag-of-words term weighting models such as BM25 or neural retrieval models based on learned sparse lexical representations such as uniCOIL. Previous experiments on the MS MARCO datasets have demonstrated the effectiveness of these methods, and they serve as baselines that are widely used by the community today. Following the recent release of the MS MARCO V2 passage and document ranking test collections, we have refreshed our doc2query and uniCOIL models. This work describes a number of resources that support competitive, reproducible baselines for both the MS MARCO V1 and V2 test collections using our Anserini and Pyserini IR toolkits. Together, they provide a solid foundation for future research on neural retrieval models using the MS MARCO datasets and beyond.

## CCS CONCEPTS

• **Information systems → Retrieval models and ranking**.

## KEYWORDS

reproducibility, neural retrieval models

**ACM Reference Format:**
Xueguang Ma, Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2022. Document Expansions and Learned Sparse Lexical Representations for MS MARCO V1 and V2. In *Proceedings of the 45th Int'l ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3477495.3531749

## 1 INTRODUCTION

The idea behind doc2query [19, 20], a form of document expansion (see Lin et al. [14], Chapter 4), is to train a (transformer-based) sequence-to-sequence model, that when given an input span of text (e.g., a passage or an extract from a longer document), generates a natural language query that the text might answer. For example, consider the following text:

> July is the hottest month in Washington DC with an average temperature of 27°C (80°F) and the coldest is January at 4°C (38°F) with the most daily sunshine hours at 9 in July. The wettest month is May with an average of 100mm of rain.

As an example, the model might generate the query "what is the weather in washington dc". Note that although the passage is clearly about weather, that particular term is not present in the text. Thus, in this case doc2query has generated a *novel* term that characterizes the content of the input text, and hence may be useful for addressing the vocabulary mismatch problem [7]. Typically, we apply inference multiple times with the same model using a top-$k$ sampling decoder to generate a set of predictions.

There are two main uses for these predicted queries (which we also call "expansions"):

(a) They can be appended to the original documents, which are then indexed as before to produce an augmented index. Retrieval proceeds exactly as before, whether in a single-stage approach or as part of a multi-stage reranking pipeline. This use case might be characterized as doc2query with "traditional" term weights, since retrieval is based on the same scoring function as before (e.g., BM25), except now over text that has been augmented with the predicted queries.
(b) They can be appended to the original documents, which together serve as input to a retrieval model that exploits learned sparse lexical representations [11]. This class of models (typically based on transformers) takes advantage of training data to *learn* term weights that are effective for retrieval. In this context, doc2query output provides a candidate set of terms that should receive non-zero weights.

It is worth noting that both use cases can take advantage of inverted indexes and query evaluation techniques that have served as the workhorses of information retrieval research and production search deployments for decades. This allows us to exploit the latest advances in pretrained transformers and other neural ranking models without abandoning mature infrastructure, since in the second use case above, we still arrive at bag-of-words representations (albeit with term weights that yield more effective retrieval than BM25). In contrast, the other main use of transformers for retrieval, so-called dense semantic retrieval models (see overview in Lin et al. [14]),

*Both authors contributed equally to this research.



*SIGIR '22, July 11–15, 2022, Madrid, Spain.*

ACM ISBN 978-1-4503-8732-3/22/07.
https://doi.org/10.1145/3477495.3531749

requires a completely separate software "stack" based on nearest neighbor search in vector space.

In terms of historical development, doc2query was first proposed in Nogueira et al. [20], but executed with "vanilla" transformers (i.e., randomly initialized, not pretrained). Nogueira and Lin [19] improved on the original instantiation by using the pretrained sequence-to-sequence model T5 [24], which yielded much higher effectiveness.[1] The T5-based variant of doc2query was originally dubbed docTTTTTquery, but is frequently written (less awkwardly) as doc2query–T5. In both cases, the models were trained on the MS MARCO passage ranking test collection.

Today, retrieval based on doc2query–T5 expansions with BM25 term weights has emerged as a standard baseline due to its balance between effectiveness and efficiency, since the approach does not require expensive neural inference at query time. This corresponds to the use case (a) above. Exploiting doc2query as an expansion component within learned sparse lexical retrieval models can be traced back to DeepImpact [16], which was subsequently improved upon in uniCOIL [12]. This corresponds to use case (b) above, and in this paper, we focus specifically on the uniCOIL model.

The primary contribution of this work is to provide a comprehensive set of resources to reproduce doc2query and uniCOIL experiments on both the V1 and V2 editions of the MS MARCO corpora. More specifically, we provide:

(1) Reproduction guides in the Anserini IR toolkit [29, 30] and implementations of self-contained one-command reproductions in the Pyserini IR toolkit [13]. We cover a wide range of experimental conditions for bag-of-words BM25, bag-of-words BM25 with doc2query–T5 expansions, and uniCOIL, encompassing every meaningful combination of corpus × model × topic set. Each experimental result we report in this paper can be reproduced with a single command-line invocation, which can be copied and pasted from our extensive online documentation.
(2) The raw corpora that underlie the implementations above in consistent and easy-to-access data formats. The doc2query–T5 expansions are packaged as Huggingface Datasets [10] and the sparse lexical representations of texts according to different uniCOIL variants are shared in a JSON line format. This means that researchers can download and manipulate these datasets using common tools, and such convenient packaging enables novel use of our results in ways we had not originally anticipated.

Since the MS MARCO V2 test collections were only released in July 2021, experimental results on these new resources have only recently become available. Many of the components described above for the V1 datasets had already existed, but in building out resources for the V2 datasets, we took the opportunity to "backport" some of our efforts to the V1 data, thus providing a consistent interface to both the V1 and V2 datasets. Together, these resources provide a solid foundation for future work on neural retrieval models, at the TREC 2022 Deep Learning Track and beyond.

[1]The curious reader might wonder: Why is Nogueira and Lin [19] a random technical report posted on a personal website that does not even appear on arXiv? The truth is that the submission was rejected as inappropriate by arXiv moderators, as the authors used the paper as a vehicle for promoting the idea of "micro-publications". Indeed, that paper holds the dubious distinction of having been rejected by arXiv.

## 2 DATA DESCRIPTION

It is hard to overstate the impact that the MS MARCO datasets have had in advancing IR and NLP research in recent years, particularly in feeding data-hungry methods based on deep learning. The origin of these datasets, prepared and released by Microsoft, date back to 2016 [17], but the most impactful has been the MS MARCO passage ranking test collection, first released in late 2018 [2]. For the first time, researchers gained access to a large collection of human-labeled (query, relevant passage) pairs to train their (neural) ranking models. The release of the dataset coincided with the advent of BERT [6], and the initial public demonstration of BERT applied to passage reranking [18] can be credited for germinating a long line of research on ranking with pretrained transformers that continues to this day [14].

### 2.1 The Original

The MS MARCO passage ranking corpus comprises 8.8m passages drawn from texts shown as "answers" at the top of Bing's search results page. The dataset contains 503k (query, relevant passage) pairs that have been labeled by human assessors, where the queries can be characterized as noisy natural language questions submitted to Bing. On average, each query has one relevant document, so these are known as "sparse judgments". Also provided are a development set of 6.9k queries and an evaluation (held-out) set of 6.8k queries. Relevance judgments are publicly available for the development set, but not the evaluation set (scores are only available via submissions to the MS MARCO leaderboard).

The MS MARCO document ranking corpus, released in 2019, comprises 3.2m full-length webpages containing the passages in the passage corpus (but crawled at a later point in time). Along with the corpus, Microsoft released training, development, and evaluation queries (367k, 5.2k, and 5.8k, respectively), where the corresponding relevance judgments were "projected" from the passage judgments (and thus are somewhat noisy).

A well-known limitation of transformer-based rerankers is that they are unable to perform inference over long segments of text all at once. One standard solution is the MaxP technique of Dai and Callan [5]: perform document retrieval, segment each document into passages, apply inference to each passage independently, then take the passage with the highest score as the representative of the document for ranking. A popular alternative is to segment the collection *prior* to indexing (i.e., each segment is indexed as a separate "document"), perform retrieval over the segments, and then feed these texts directly to a transformer-based reranker.

As we use the latter approach for many of our group's work on neural retrieval models, we have created a *segmented* variant of the MS MARCO document corpus. Segments were created by applying a 10-sentence sliding window with a 5-sentence stride; experiments reported in Pradeep et al. [22] have demonstrated this to be an effective approach, hitting a "sweet spot" in providing sufficient context for downstream rerankers but not overwhelming them with texts that are too long.

The MS MARCO passage and document ranking test collections were also used for the Deep Learning Tracks at TREC 2019 [4] and TREC 2020 [3]. The resources we provide in this paper cover topic sets from those two evaluations as well.

## 2.2 The Sequel

In July 2021, Microsoft released what can be characterized as a refresh of the MS MARCO collections. These are referred to as the "Version 2" (V2) collections, and to reduce ambiguity, we refer to the original MS MARCO collections as the V1 editions. Unlike in the V1 datasets, where the passage and document corpora were gathered at different points in time, the MS MARCO V2 passage and document corpora were created from the same underlying Bing web crawl. The V2 passage corpus comprises 138.4m passages and comes in two "flavors", both distributed in a JSON lines format:

- *Passage corpus.* This is the "default" version of the corpus, where each record contains a `passage` field with the passage text, along with a pointer to the document in the document corpus it was extracted from.
- *Augmented passage corpus.* The "default" passage corpus above is missing additional information such as the title and URL of the source page. This information is available in the document corpus (see below), and explicitly "projected" back into the passage corpus. In the augmented passage corpus, each record contains three additional fields: `url`, `title`, and `headings`.

The V2 document corpus comprises 12.0m documents extracted from the source webpages of the passages from the passage corpus. The organizers also curated two different "flavors", both distributed in JSON lines format:

- *Document corpus.* This is the "default" version of the corpus. In addition to the `body` field containing the body text, each record contains the following fields: `url`, `title`, and `headings`. The first two are self-explanatory; the last field is a concatenation of all the headings identified in the source webpage.
- *Segmented document corpus.* For the same reasons as with the V1 document corpus discussed above, the organizers created a *segmented* version of the document corpus. Each document was trimmed to 10k characters, from which segments were generated by applying a 10-sentence sliding window with a 5-sentence stride. Each record retains the `url`, `title`, and `headings` fields from the full document. The process for generating these segments is similar to the procedure we used for the V1 document corpus, except that the V2 segmented document corpus is an "official release" by the organizers. The standardization of the segments makes it easier to compare downstream techniques.

Overall, the MS MARCO V2 corpora are larger, cleaner, and more consistently organized than the V1 corpora.

Along with the release of the corpora, Microsoft also shared training data as well as two development sets each, for passages and documents. Furthermore, these resources were also used for the Deep Learning Track at TREC 2021. These new datasets promise to continue driving advances in neural ranking models moving forward, and to help, our work covers all these different corpus variants and topic sets.

# 3 METHODS

In this section, we describe the two core technical components that underlie the resources we have assembled: doc2query and uniCOIL.

## 3.1 doc2query

One key strength of the doc2query method is its simplicity, both conceptually and in terms of implementation. The model can be straightforwardly trained with existing off-the-shelf sequence-to-sequence neural toolkits with minimal modifications. The original method was described in Nogueira et al. [20] using "vanilla" (i.e., randomly initialized) transformers. The obvious improvement was to use a pretrained model, which was explored in Nogueira and Lin [19] using T5 [24], creating the doc2query–T5 model. We build on that work here, but in all cases, our models are relatively simple to train using Google's T5 implementation.[2]

The original doc2query–T5 model, as described in Nogueira and Lin [19], was trained on the MS MARCO V1 passage ranking dataset. Under a standard sequence-to-sequence formulation, passages from the training set are fed to the model, which learns to generate the corresponding query. The T5-base model was trained with a constant learning rate of $10^{-3}$ for 4k iterations with batches of 256, which corresponds to 2 epochs on the MS MARCO training set. The model used a maximum of 512 input tokens and 64 output tokens; none of the MS MARCO training passages or queries had to be truncated to meet these length restrictions. Similar to Nogueira et al. [20], experiments showed that the top-$k$ sampling decoder produced higher-quality queries than beam search (as measured by retrieval effectiveness).

Model training and inference were performed using Google's TPU v3-8s. Training took less than 1.5 hours on a single TPU. We have made this model available on Huggingface's model hub, and as of April 2022, it receives over 4k downloads per month. For inference, we used the setting of $k = 10$ to generate 40 queries per passage; we fed the model just the passage text. Generating expansions for all 8.8m passages in the MS MARCO V1 passage corpus required approximately 320 hours on a single TPU. Note that inference is trivially parallelizable and linear with respect to the number of samples.

To generate expansions, doc2query can be directly applied to MS MARCO V1 passages. However, for longer documents, and specifically the V1 document corpus, it is not practical (both from the efficiency and quality perspectives) to directly apply doc2query to an entire document at once. To address this issue, we applied expansions to the segments from the V1 segmented document corpus, feeding to the model independently each text segment, prepended with the title of the document it came from. Due to the larger dataset sizes involved, we only sampled 10 queries per segment. The expansions can be used to augment both the original corpus and the segmented corpus; we describe details in Section 5.1. Note that this approach allowed us to reuse models trained on passages directly on documents; that is, we did not separately train a doc2query model for documents.

For the MS MARCO V2 corpora, we trained another T5-base model in a similar fashion, with a constant learning rate of $10^{-3}$ for 4k iterations with batches of 256. This corresponds to roughly 4 epochs with the MS MARCO V2 passage training set.

At inference time, we used the top-$k$ ($k = 10$) sampling decoder to sample 20 queries per passage for the passage corpus and 10 queries per segment for each segment from the segmented document corpus.

[2] https://github.com/google-research/text-to-text-transfer-transformer

Again, as with the V1 data, for the document condition, we fed the combination of document title and segment text to the model trained on the passage data. We reduced the number of queries sampled from 40 to 20 in the V2 passage corpus since it is roughly 16 times larger than the V1 passage corpus. As a reference, we performed the expansions using preemptible TPU v3-8s, which took around 2500 hours and cost us roughly $6k USD for the MS MARCO V2 passage corpus.

### 3.2 uniCOIL

Our uniCOIL model [12] is a variant of the COIL-tok model [8], which follows a basic bi-encoder design, just like dense retrieval models such as DPR [9] and ANCE [27]. The major difference is that the model uses exact lexical match to compute the similarity between queries and passages instead of performing matching in a latent semantic space. That is, the basis of the vector representation generated by uniCOIL is the vocabulary space.

Given a passage $p$, COIL-tok computes the embedding vector $v_i^p$ of the $i$-th token $p_i$ by:

$$v_i^p = W \cdot \text{BERT}(d)[i]$$

where $\text{BERT}(d)[i]$ is the output of the last layer of BERT corresponding to the $i$-th token and $W$ is a projection from 768 dimensions to 32 dimensions. The similarity score between a query and a passage is computed by:

$$\text{Sim}(q,p) = \sum_{q_j \in q \cap p} \max_{p_i = q_j} < v_j^q, v_i^p >$$

During model training, the similarity between queries and relevant passages is optimized according to NCE loss.

One limitation of COIL-tok is that the model requires custom search infrastructure for efficient top-$k$ retrieval since the representation of each token is a 32-dimensional vector. The primary contribution of uniCOIL over COIL-tok is a modification that enables the use of standard inverted indexes for efficient retrieval over representations generated by the model, which can be accomplished with only a modest loss of effectiveness compared to the full 32-dimensional representations from COIL-tok. Thus, retrieval with uniCOIL fits seamlessly into our existing Anserini and Pyserini infrastructure built on Lucene.

To accomplish this, uniCOIL changes the projection matrix $W$ to a 768-to-1 transformation and further adds a ReLU operation. Thus, instead of representing each token as a vector, uniCOIL assigns each token a positive scalar weight, and query–passage similarity becomes:

$$\text{Sim}(q,p) = \sum_{q_j \in q \cap p} \max_{p_i = q_j} w_j^q \cdot w_i^p$$

which is equivalent to:

$$\text{Sim}(q,p) = \sum_{j=1}^{n} f(q_j) \cdot g(q_j, p)$$

$$\text{where } f(q_j) = w_j^q$$

$$g(q_j,p) = \begin{cases} \max_{p_i = q_j} w_i^p & \text{if } q_j \in p \\ 0 & \text{otherwise} \end{cases}$$

Note that the similarity equation takes exactly the same functional form as top-$k$ retrieval with "traditional" bag-of-words models such as BM25 (i.e., an inner product), but now, the sparse lexical representations of queries and passages are learned by deep neural networks. By further quantizing the token weights into integers, these become impact scores [1], which are directly compatible with inverted indexes and standard query evaluation algorithms.

Despite learning better term weights, COIL-tok nevertheless relies on exact matching, and thus still suffers from the vocabulary mismatch problem [7], where relevant documents and queries use different terms to express the same concepts. With uniCOIL, we address this issue by performing document expansion to increase the chances of exact term matches. This is where doc2query–T5 comes in: we augment the passages by appending terms from the predicted queries that do not appear in the original text. This can be viewed as generating for uniCOIL a set of candidate terms that should receive non-zero weights in the final representation; on this set of terms, uniCOIL then learns appropriate weights.

In this paper, we describe experiments on MS MARCO V1 and V2 corpora with two separate models:

- uniCOIL: our "default" model is trained on the MS MARCO V1 passage data, augmented with novel terms from 40 predicted queries generated by doc2query–T5.
- uniCOIL with no expansions (noexp): as an ablation condition, we trained a variant model on the MS MARCO V1 passage data, but without any expansions.

Comparing these two models allows us to separately quantify the impact of better term weighting and document expansion.

Both models are trained with the same hyperparameter settings, with a learning rate of $5 \times 10^{-6}$ for 5 epochs with batches of 8. We used the `bert-base-uncased` model from Huggingface as our starting backbone. Training examples in each batch comprise 1 positive passage and 7 negative passages, where the negative passages are sampled with replacement from the top-1000 BM25 hard negative passages for each batch. It takes around 12 hours to train a model on a single V100 GPU. The code to train uniCOIL is shared in the COIL repository on GitHub.[3] Both models are available on Huggingface's model hub, and as of April 2022, together they receive approximately 1k downloads per month.

Applying uniCOIL for passage ranking (on both the V1 and V2 corpora) is straightforward. We apply inference over all passages (specifically, just the passage text) to generate the sparse lexical representations. These are encoded in JSON lines format (more details below), and can be directly indexed by Anserini or Pyserini. Note that we decided *not* to train a separate model for the V2 passage corpus and instead applied the uniCOIL model trained on the V1 data in a zero-shot manner.

For document ranking, the same length limitations with transformer models also apply, since uniCOIL is still based on BERT. Here, we adopt a standard solution: inference is applied to the *segmented* versions of the corpora (both V1 and V2). In both cases, we feed to the model the segment text prepended with the document title (an earlier experiment confirmed that the addition of the document title improved effectiveness). The representations of

[3] https://github.com/luyug/COIL/tree/main/uniCOIL

each segment are indexed separately, and at retrieval time the top-scoring segment for each document is taken as its representative to generate a document ranking.

## 4 AVAILABLE RESOURCES

As part of this work, we make two broad categories of resources available for working with the MS MARCO V1 and V2 collections: (1) reproduction guides and implementations of one-command reproduction, and (2) raw corpora that underlie the implementations above in consistent and easy-to-access data formats.

### 4.1 Reproduction Guides and Commands

As discussed above, one main advantage of doc2query and uniCOIL is that they can improve retrieval effectiveness *without abandoning inverted indexes* and associated "mature" infrastructure that has been refined and optimized over several decades. To demonstrate this, we provide a number of reproduction guides in our Anserini IR toolkit [29, 30], which is built around the open-source Lucene search library written in Java.

For doc2query, our existing indexing and retrieval pipeline did not have to change at all, as document expansion occurs in a data preparation stage and retrieval remains based on BM25 scoring. For uniCOIL, we take advantage of the "pseudo-document trick". In Anserini, there is an abstraction for a "generator" that takes a raw document representation (e.g., in JSON) to produce the actual text to be indexed. For uniCOIL and other learned sparse lexical retrieval models, we have implemented a generator that creates a "fake document", where each term is repeated the same number of times as its (quantized) integer weight. For example, if term *A* receives a weight of 5, we simply repeat *A* five times in the pseudo-document. Once fed into the standard Anserini indexing pipeline, the integer weight (i.e., its impact score) is stored in the term frequency position of a posting. For retrieval, we implemented a scoring function that sums term frequencies (which are the impact scores) of matching terms, which generates exactly the rankings we desire. In Anserini, we rely on queries that have already been pre-encoded, but for Pyserini (more below), neural inference on the queries can be applied "on the fly".

Our reproduction guides are linked from the main landing page of the Anserini code repository on GitHub.[4] For MS MARCO V1, we have documentation devoted to the following combinations of corpus, model, and topic set:

- corpus: { passage, document, segmented document } ×
- model: { BM25, BM25 (with doc2query–T5), uniCOIL (noexp), uniCOIL (with doc2query–T5) } ×
- topic set: { MS MARCO dev, TREC 2019, TREC 2020 }.

For MS MARCO V2, we have documentation devoted to the following combinations of corpus, model, and topic set:

- corpus: { passage, augmented passage, document, segmented document } ×
- model: { BM25, BM25 (with doc2query–T5), uniCOIL (noexp), uniCOIL (with doc2query–T5) } ×
- topic set: { MS MARCO dev, MS MARCO dev2, TREC 2021 }.

[4] http://anserini.io/

Our documentation exhaustively covers the entire experimental matrix for all possible combinations.[5] These experiments are grouped at a high level into a number of reproduction guides, which are written to explicitly include the exact command-line invocations necessary to generate the experimental results (i.e., those we report in Section 5). These guides cover the entire "lifecycle" of an IR experiment, including building the inverted index, performing retrieval runs, and computing evaluation metrics. The practical upshot is that anyone can reproduce our experiments by copying and pasting a sequence of commands into a shell.

While our reproduction guides are meant for human consumption and appear as static pages in our code repository, in reality, they are automatically generated from underlying YAML configuration files as part of our regression framework.[6] These configuration files specify the location of raw corpora on the file systems of our research group's servers and specify parameters for executing every step in the lifecycle of an IR experiment. Furthermore, these files encode data to validate output at each stage, for example, the number of documents that should appear in an index and the expected scores of a particular retrieval run. For each experimental condition, a Python script assembles the configuration information into command-line invocations to build the inverted index, verify index integrity, perform retrieval, evaluate results, validate scores, and build the human-readable reproduction guides. The complete regression test suite in Anserini contains over 150 such individual end-to-end tests (as of April 2022), which we run on a regular basis to ensure that code changes do not introduce any bugs.

In our relentless quest to "reduce friction" and lower barriers to reproducibility, we felt that the above resources could be further improved and that the process of reproducing experimental results could be further simplified. Our ideal can be characterized as one-command reproduction: everything that's necessary should be self-contained in an easily shareable computational artifact.

This is accomplished via Pyserini, our Python toolkit for reproducible information retrieval research with sparse and dense representations [13]. Python has evolved into the lingua franca of IR research today, given the dominance of neural methods and the fact that popular deep learning toolkits such as PyTorch and TensorFlow have adopted Python as their frontend language. As such, we believe that Python, via packages shared on the Python Package Index (PyPI), provides the broadest reach.

In Pyserini, the experimental results reported in Section 5 can be reproduced with a single command. For example, on the MS MARCO V1 passage corpus, performing BM25 ranking on doc2query expansions:

```
python -m pyserini.search.lucene \
  --index msmarco-v1-passage-d2q-t5 \
  --topics msmarco-passage-dev-subset \
  --output run.msmarco-passage.expanded.txt \
  --output-format msmarco \
  --bm25
```

And as another example, applying our uniCOIL model to the MS MARCO V2 passage corpus, running on the MS MARCO V2 development queries:

[5] Some combinations are not possible; for example, uniCOIL only works on document segments.

[6] https://github.com/castorini/anserini/blob/master/docs/regressions.md

```
python -m pyserini.search.lucene \
  --index msmarco-v2-passage-unicoil-0shot \
  --topics msmarco-v2-passage-dev \
  --encoder castorini/unicoil-msmarco-passage \
  --output run.msmarco-v2-passage.unicoil.txt \
  --batch 144 --threads 36 \
  --hits 1000 \
  --impact
```

All documentation is linked from the main landing page of the Pyserini code repository on GitHub.[7] In both these examples, the main driver program `pyserini.search.lucene` dispatches queries, via a series of wrapper classes, to Lucene, gathers the results, and writes the output to a file. The entire implementation is self-contained in that the driver program "knows" where to find all the components necessary to generate the specific run. For example, we share prebuilt indexes for all the corpora in Section 4.2; the driver program downloads the indexes on the first invocation and then caches them for subsequent reuse. Similarly, all the topic sets mentioned in Section 4.2 are already packaged in Pyserini, so the user does not need to separately download them. Finally, in the case of uniCOIL, we point the driver program at a query encoder hosted on Huggingface's model hub, which is downloaded and used to generate the appropriate query representations.

The upshot is that for every experimental condition in our reproduction guides (i.e., corpus × model × topics set matrix), there is a corresponding reproduction command that is wholly self-contained. That is, copying and pasting the correct invocation in a command shell will generate the results presented in Section 5.

## 4.2 Raw Corpora

In addition to the reproduction guides and one-command reproductions described above, we also make publicly available the raw corpora that underlie all the experiments described in this paper in consistent and easy-to-access data formats. Researchers can download and manipulate these resources using common tools, and we hope that such convenient packaging enables novel use of our results in ways we had not originally anticipated.

The expansions themselves for both MS MARCO V1 and V2 are available as Huggingface Datasets [10]. The Datasets package provides one-line dataloaders, a unified interface for downloading and manipulating the data, and simplified integration with Huggingface Transformers [26]. By packaging our expansions in this way, we get all the benefits of the Datasets API "for free".[8] For example, the dataset preview feature on the landing pages corresponding to our datasets makes it easier for researchers to first inspect our expansions prior to downloading the associated files onto their own systems. The popularity of Huggingface greatly increases the reach of our resources.

Specifically, we provide the following datasets, all under the `castorini` group on Huggingface:

(1) `msmarco_v1_passage_doc2query-t5_expansions`: all expansions generated for the MS MARCO V1 passage corpus.
(2) `msmarco_v1_doc_segmented_doc2query-t5_expansions`: all expansions generated for the MS MARCO V1 segmented document corpus.
(3) `msmarco_v1_doc_doc2query-t5_expansions`: all expansions generated for the MS MARCO V1 document corpus. Note that this is just a repackaged version of (2), where we gather together for each document all expansions from each segment comprising that document.
(4) `msmarco_v2_passage_doc2query-t5_expansions`: all expansions generated for the MS MARCO V2 passage corpus.
(5) `msmarco_v2_doc_segmented_doc2query-t5_expansions`: all expansions generated for the MS MARCO V2 segmented document corpus.
(6) `msmarco_v2_doc_doc2query-t5_expansions`: all expansions generated for the MS MARCO V2 document corpus. Note that this is just a repackaged version of (5), similar to (3).

In addition to the doc2query–T5 expansions, we also provide the raw sparse lexical representations of the different corpora based on our uniCOIL model variants. These are the results of feeding each document to the transformer model: the output, for each input segment of text, is a mapping from terms to weights. This can be straightforwardly captured as JSON maps, and thus we have adopted a JSON lines format for distributing these datasets.

Specifically, we provide eight different corpora, covering uniCOIL (with doc2query–T5) and uniCOIL (noexp) on the V1 passage corpus, the V1 segmented document corpus, the V2 passage corpus, the V2 segmented document corpus. Instructions for downloading these raw corpora can be found on the reproduction pages associated with our experimental configurations.

# 5 RESULTS

In this section, we overview results for every meaningful combination of corpus × model × topic set described in Section 4. There are too many experimental conditions to describe the exact configuration of each, and thus we refer the reader to our extensive online documentation, which details the parameter settings and provides the command-line invocations necessary to generate the figures reported in each cell of our results tables. Links to these documentation pages can be found on the main landing page of the Anserini and Pyserini GitHub repositories.

For each topic set, we include the official metric, e.g., reciprocal rank (RR) on the MS MARCO development queries and nDCG for the TREC topics, as well as other common metrics such as average precision (AP) and recall at a cutoff of 1000 hits (R@1K). The latter is frequently used to provide an upper bound on end-to-end effectiveness when an approach is used as a first-stage retriever to feed downstream rerankers. To make the results deterministic, we report figures from Anserini, which uses pre-encoded queries where neural inference has already been applied. Pyserini is able to apply query inference "on the fly", but the effectiveness results may be slightly different.

Due to the large number of experimental conditions, we deliberately dispense with significance testing on metric differences, especially mindful of the pitfalls associated with multiple hypothesis testing. Our work here exists primarily as a resource to "tie everything together", as the impact of individual innovations has

[7] http://pyserini.io/
[8] Prior to this reorganization, we made V1 expansions available as flat text files, organized in a way that was difficult to manipulate.

| | | TREC 2019 DL Track | | | TREC 2020 DL Track | | | Dev | |
|---|---|---|---|---|---|---|---|---|---|
| | **Condition** | AP | nDCG@10 | R@1K | AP | nDCG@10 | R@1K | RR@10 | R@1K |
| (1a) | BM25 | 0.3013 | 0.5058 | 0.7501 | 0.2856 | 0.4796 | 0.7863 | 0.1840 | 0.8526 |
| (1b) | BM25+RM3 | 0.3390 | 0.5180 | 0.7998 | 0.3019 | 0.4821 | 0.8217 | 0.1564 | 0.8606 |
| (2a) | BM25 (w/ doc2query–T5) | 0.4034 | 0.6417 | 0.8310 | 0.4074 | 0.6187 | 0.8452 | 0.2723 | 0.9470 |
| (2b) | BM25+RM3 (w/ doc2query–T5) | 0.4485 | 0.6548 | 0.8861 | 0.4295 | 0.6172 | 0.8699 | 0.2140 | 0.9463 |
| (3a) | uniCOIL (noexp) | 0.4033 | 0.6433 | 0.7752 | 0.4021 | 0.6523 | 0.7861 | 0.3153 | 0.9239 |
| (3b) | uniCOIL (w/ doc2query–T5) | 0.4612 | 0.7024 | 0.8292 | 0.4430 | 0.6745 | 0.8430 | 0.3516 | 0.9582 |
| (4) | uniCOIL (w/ doc2query–T5) + TCT-ColBERTv2 | 0.4750 | 0.7166 | 0.8533 | 0.5004 | 0.7187 | 0.8665 | 0.3764 | 0.9761 |
| (5) | TREC Best | 0.5030 | 0.7645 | - | 0.5445 | 0.8031 | - | - | - |

**Table 1: Results on the MS MARCO V1 passage corpus.**

| | | TREC 2019 DL Track | | | TREC 2020 DL Track | | | Dev | |
|---|---|---|---|---|---|---|---|---|---|
| | **Condition** | AP@100 | nDCG@10 | R@1K | AP@100 | nDCG@10 | R@1K | RR@100 | R@1K |
| (1a) | BM25 [original] | 0.2434 | 0.5176 | 0.6966 | 0.3793 | 0.5286 | 0.8085 | 0.2299 | 0.8856 |
| (1b) | BM25 [segmented] | 0.2449 | 0.5302 | 0.6871 | 0.3586 | 0.5281 | 0.7755 | 0.2684 | 0.9178 |
| (1c) | BM25+RM3 [original] | 0.2774 | 0.5170 | 0.7503 | 0.4014 | 0.5225 | 0.8257 | 0.1622 | 0.8791 |
| (1d) | BM25+RM3 [segmented] | 0.2884 | 0.5764 | 0.7384 | 0.3774 | 0.5179 | 0.8041 | 0.2412 | 0.9355 |
| (2a) | BM25 (w/ doc2query–T5) [original] | 0.2700 | 0.5968 | 0.7190 | 0.4230 | 0.5885 | 0.8403 | 0.2880 | 0.9259 |
| (2b) | BM25 (w/ doc2query–T5) [segmented] | 0.2798 | 0.6119 | 0.7165 | 0.4150 | 0.5957 | 0.8046 | 0.3179 | 0.9490 |
| (2c) | BM25+RM3 (w/ doc2query–T5) [original] | 0.3045 | 0.5897 | 0.7738 | 0.4229 | 0.5407 | 0.8596 | 0.1831 | 0.9128 |
| (2d) | BM25+RM3 (w/ doc2query–T5) [segmented] | 0.3021 | 0.6297 | 0.7481 | 0.4268 | 0.5850 | 0.8270 | 0.2818 | 0.9547 |
| (3a) | uniCOIL (noexp) | 0.2665 | 0.6349 | 0.6391 | 0.3698 | 0.5893 | 0.7623 | 0.3409 | 0.9420 |
| (3b) | uniCOIL (w/ doc2query–T5) | 0.2789 | 0.6396 | 0.6652 | 0.3882 | 0.6033 | 0.7869 | 0.3531 | 0.9546 |
| (4) | uniCOIL (w/ doc2query–T5) + TCT-ColBERT-V2 | 0.2961 | 0.6709 | 0.6964 | 0.4295 | 0.6358 | 0.8146 | 0.3827 | 0.9684 |
| (5) | TREC Best | 0.3137 | 0.7257 | - | 0.5422 | 0.6934 | - | - | - |

**Table 2: Results on the MS MARCO V1 document corpus.**

been separately explored in other papers. Thus, for presentational clarity, we focus on the "big picture".

Two other caveats are worth noting: First, our resources are meant to serve as competitive baselines to support future work. We provide figures from state-of-the-art methods only as points of reference. Second, all our conditions represent *single-stage* retrieval. Our methods can serve as the first-stage retriever in multi-stage ranking architectures to achieve even higher effectiveness.

## 5.1 MS MARCO V1

Results on the MS MARCO V1 collections are shown in Table 1 for passage ranking and in Table 2 for document ranking. All runs here retrieved 1000 hits per query.

**Passage Ranking.** Looking at the passage ranking results, rows (1a) and (2a) illustrate the impact of doc2query–T5 expansions (both with BM25). We note metric improvements across the board, which is consistent with previous results demonstrating the effectiveness of doc2query [19, 20]. In rows (1b) and (2b), we add RM3 query expansion on top of BM25 [31]. On the TREC data, we see improvements in recall-oriented metrics but at best marginal gains in nDCG@10. On the MS MARCO development queries, reciprocal rank actually goes down, but this is an artifact of sparse judgments; these effects are well known and discussed in Lin et al. [14].

In Table 1, rows (3a) and (3b) present an interesting contrast: the uniCOIL model with and without doc2query–T5 expansions (see Section 3.2). The first case (noexp) can be colloquially understood as "like BM25", except the term weights are learned: uniCOIL assigns zero weights to terms that do not appear in the document. Thus, comparing row (1a) to row (3a), we can quantify the impact of *learning* term weights from large amounts of manually labeled data. No surprise, we see increases in effectiveness across the board.

The contrast between rows (3b) and (2b) explores the role of term weighting and document expansion based on doc2query–T5. In particular, we see that, on the TREC data, uniCOIL (w/ doc2query–T5), row (3b), is more effective than BM25+RM3 (w/ doc2query–T5), row (2b), although recall of the first appears lower on TREC topics. BM25+RM3 is a two-stage retrieval approach, since query expansion terms are identified based on initial search results, whereas with uniCOIL (w/ doc2query–T5) the passage representations are computed in a query-independent manner. These results show that BM25+RM3 remains a competitive non-neural baseline.

In row (4), we combine the results from row (3b), uniCOIL (w/ doc2query–T5), which represents the most effective sparse lexical retrieval model here, with results from our TCT-ColBERTv2 dense retrieval model [15]. These sparse–dense hybrid results were generated by a linear combination of normalized scores from both, with the weights tuned on training data. The goal here is to show that dense retrieval models provide additional relevance signals,

but at the cost of requiring an independent "infrastructure stack", since top-$k$ retrieval using dense representations cannot be accomplished with inverted indexes. The TCT-ColBERTv2 model is competitive, but still behind the state of the art, which suggests that there remains more headroom to further increase effectiveness in a single-stage retrieval setup.

Finally, in row (5) we report results of the best run from TREC in terms of nDCG@10. The best score on the MS MARCO development queries can be found on the MS MARCO leaderboard.[9] We simply invite interested readers to consult the leaderboard, since the state of the art changes with some regularity and any figure we copy here will become outdated. It is no surprise that our sparse–dense hybrid still lags behind the best TREC submissions in 2019 [28] and 2020 [23]. However, this direct comparison is not fair because both of those runs used expensive neural rerankers that incur much higher latency compared to our single-stage approach.

**Document Ranking.** Results for document ranking are shown in Table 2. Recall from Section 4.2 that there are two "flavors" of the document corpus: the original version and the segmented version. With the original corpus, the entirety of each document comprises the text to be indexed. With the segmented corpus, each segment is prepended with the title of the document it comes from, and together this forms the unit of indexing. For retrieval on the segmented corpus, we follow the standard approach of taking the highest scoring passage from each document as its representative to generate a document ranking.

Retrieval effectiveness on these two different corpus "flavors" is contrasted in row (1a) vs. row (1b). On the MS MARCO development queries, the segmented approach appears to be better, but we suspect that this is an artifact of the relevance judgments. Since they were "projected" from passage judgments, metrics likely favor techniques that focus on passages. On the TREC data, both approaches are comparable. In rows (1c) and (1d), we add RM3 query expansion on top of the two base BM25 models. Improvements from query expansion are expected and consistent with the literature. Again, there does not appear to be a consistent winner among the corpus "flavors" based on TREC data.

For doc2query–T5 expansions with BM25 term weighting, we have the option of using the original corpus or the segmented corpus as the starting point. In the first case, we gather all expansions from each segment in the document and concatenate them all to the original document. In the second case, the expansions for each segment are concatenated to that segment and treated independently (again, each segment comprises the segment text prepended with the document title). Comparing rows (2a) and (2b), we see that the segmented variant achieves higher scores on the development queries (once again, likely an evaluation artifact), but results are comparable for both corpus "flavors" on the TREC data. Additional query expansion with RM3, rows (2c) and (2d), further improves effectiveness, which is expected. Based on nDCG@10, it appears that the segmented version is more effective.

Results with uniCOIL are shown in rows (3a) and (3b). As discussed in Section 3.2, it is difficult to apply uniCOIL directly to full-length documents. Instead, we apply inference to the segmented document corpus; each segment is prepended with the document title. At retrieval time, the highest scoring segment from each document is used for ranking. Comparing row (3a) to row (1b), we clearly see the benefits of learning term weights from large amounts of supervised data. Document expansion with doc2query–T5 further improves uniCOIL effectiveness, although the gains are more modest compared to results on the passage corpus.

Interestingly, on TREC data, uniCOIL (w/ doc2query–T5), row (3b), is not consistently more effective than the best BM25+RM3 (w/ doc2query–T5) condition, row (2d). In fact, looking at average precision, uniCOIL scores are lower across both sets of TREC topics. This contrasts with the comparable conditions for passage ranking, which suggests that document ranking presents a more difficult task for neural models overall. Since techniques we have implemented to handle long texts are quite simple (based on segments that are processed independently), this finding is perhaps not surprising and points to a future research direction.

Row (4) represents our sparse–dense hybrid and row (5) shows the best TREC runs. Again, as with passage ranking, we invite the reader to consult the leaderboard[10] for the latest state of the art on the development queries. Compared to the best TREC submissions in 2019 [28] and 2020 [21], we arrive at a similar conclusion as with passage ranking: our sparse–dense hybrid remains behind in terms of effectiveness but is faster at inference time since we do not require an expensive neural reranking stage.

## 5.2 MS MARCO V2

Results on the MS MARCO V2 collections are shown in Table 3 for passage ranking and in Table 4 for document ranking, organized in a similar manner as the V1 collections. All runs here retrieved 1000 hits per query, although note that the TREC 2021 guidelines called for returning only 100 hits per query.

The initial release of the MS MARCO V2 datasets came with two sets of development queries (dev and dev2); these are sparse (and highly incomplete) judgments, and interpreting the results requires nuance. Furthermore, Voorhees et al. [25] identified a number of problematic issues with the test collection built from the TREC 2021 Deep Learning Track: Due to score saturation effects (i.e., the best systems obtain very high scores), the evaluation instrument is unreliable. Additionally, standard diagnostics show that existing judgments are highly incomplete, and thus the collection is not reusable (i.e., cannot be used to fairly evaluate systems that did not participate in the original track). Results presented here should be interpreted with these caveats in mind, and our discussions focus primarily on interesting differences from the V1 results.

**Passage Ranking.** For the passage ranking task, we ran baseline bag-of-words techniques (BM25 and BM25+RM3) on the two "flavors" of the passage corpus (original and augmented). These are shown in rows (1a) to (1d). With the augmented corpus, all available fields are concatenated together and fed to the indexer. On the dev and dev2 queries, results on the augmented version are clearly better (in terms of RR@100), but on the TREC data, effectiveness on the augmented passage corpus is lower. This contrast appears with the doc2query–T5 expansions as well, shown in rows (2a) to (2d):

[9] https://microsoft.github.io/MSMARCO-Passage-Ranking-Submissions/leaderboard/

[10] https://microsoft.github.io/MSMARCO-Document-Ranking-Submissions/leaderboard/

| | | TREC 2021 Deep Learning Track | | | | Dev | | Dev2 | |
|---|---|---|---|---|---|---|---|---|---|
| | **Condition** | AP | nDCG@10 | RR@100 | R@100 | RR@100 | R@1K | RR@100 | R@1K |
| (1a) | BM25 [original] | 0.1357 | 0.4458 | 0.5060 | 0.3261 | 0.0719 | 0.5733 | 0.0802 | 0.5839 |
| (1b) | BM25 [augmented] | 0.0977 | 0.3977 | 0.5303 | 0.2709 | 0.0872 | 0.6925 | 0.0917 | 0.6933 |
| (1c) | BM25+RM3 [original] | 0.1632 | 0.4480 | 0.4925 | 0.3498 | 0.0619 | 0.5933 | 0.0654 | 0.6049 |
| (1d) | BM25+RM3 [augmented] | 0.1050 | 0.3906 | 0.4906 | 0.2795 | 0.0674 | 0.6859 | 0.0701 | 0.6838 |
| (2a) | BM25 (w/ doc2query–T5) [original] | 0.1874 | 0.4816 | 0.6848 | 0.4076 | 0.1072 | 0.7083 | 0.1123 | 0.7151 |
| (2b) | BM25 (w/ doc2query–T5) [augmented] | 0.1649 | 0.4702 | 0.6391 | 0.3883 | 0.1172 | 0.7647 | 0.1170 | 0.7659 |
| (2c) | BM25+RM3 (w/ doc2query–T5) [original] | 0.2271 | 0.5099 | 0.6651 | 0.4444 | 0.0948 | 0.7174 | 0.0985 | 0.7240 |
| (2d) | BM25+RM3 (w/ doc2query–T5) [augmented] | 0.1932 | 0.4834 | 0.5882 | 0.4295 | 0.0880 | 0.7625 | 0.0887 | 0.7649 |
| (3a) | uniCOIL (noexp) | 0.2193 | 0.5756 | 0.6991 | 0.4246 | 0.1342 | 0.7010 | 0.1385 | 0.7114 |
| (3b) | uniCOIL (w/ doc2query–T5) | 0.2538 | 0.6159 | 0.7311 | 0.4731 | 0.1499 | 0.7616 | 0.1577 | 0.7671 |
| (4) | uniCOIL (w/ doc2query–T5) + TCT-ColBERTv2 | 0.2549 | 0.6259 | 0.7759 | 0.4839 | 0.1904 | 0.8646 | 0.1939 | 0.8580 |
| (5a) | TREC Best (overall) | 0.3193 | 0.7494 | 0.8732 | - | - | - | - | - |
| (5b) | TREC Best (single-stage) | 0.2842 | 0.6714 | 0.8045 | - | - | - | - | - |

**Table 3: Results on the MS MARCO V2 passage corpus.**

| | | TREC 2021 Deep Learning Track | | | | Dev | | Dev2 | |
|---|---|---|---|---|---|---|---|---|---|
| | **Condition** | AP | nDCG@10 | RR@100 | R@100 | RR@100 | R@1K | RR@100 | R@1K |
| (1a) | BM25 [original] | 0.2126 | 0.5116 | 0.8367 | 0.3195 | 0.1572 | 0.8054 | 0.1659 | 0.8029 |
| (1b) | BM25 [segmented] | 0.2436 | 0.5776 | 0.8937 | 0.3478 | 0.1896 | 0.8542 | 0.1930 | 0.8549 |
| (1c) | BM25+RM3 [original] | 0.2453 | 0.5339 | 0.7994 | 0.3374 | 0.0974 | 0.7694 | 0.1028 | 0.7736 |
| (1d) | BM25+RM3 [segmented] | 0.2933 | 0.6185 | 0.9018 | 0.3892 | 0.1664 | 0.8605 | 0.1701 | 0.8633 |
| (2a) | BM25 (w/ doc2query–T5) [original] | 0.2387 | 0.5792 | 0.8866 | 0.3443 | 0.2011 | 0.8614 | 0.2012 | 0.8568 |
| (2b) | BM25 (w/ doc2query–T5) [segmented] | 0.2683 | 0.6289 | 0.9454 | 0.3656 | 0.2226 | 0.8982 | 0.2234 | 0.8952 |
| (2c) | BM25+RM3 (w/ doc2query–T5) [original] | 0.2608 | 0.5392 | 0.8342 | 0.3580 | 0.1144 | 0.8189 | 0.1169 | 0.8245 |
| (2d) | BM25+RM3 (w/ doc2query–T5) [segmented] | 0.3192 | 0.6555 | 0.8960 | 0.4119 | 0.1974 | 0.9000 | 0.1966 | 0.8967 |
| (3a) | uniCOIL (noexp) | 0.2587 | 0.6495 | 0.9282 | 0.3563 | 0.2231 | 0.8987 | 0.2314 | 0.8995 |
| (3b) | uniCOIL (w/ doc2query–T5) | 0.2718 | 0.6783 | 0.9684 | 0.3700 | 0.2419 | 0.9122 | 0.2445 | 0.9172 |
| (4) | uniCOIL (w/ doc2query–T5) + TCT-ColBERTv2 | 0.3041 | 0.7022 | 0.9576 | 0.3893 | 0.2740 | 0.9305 | 0.2707 | 0.9366 |
| (5a) | TREC Best (overall) | 0.3111 | 0.7437 | 0.9795 | - | - | - | - | - |
| (5b) | TREC Best (single-stage) | 0.2887 | 0.7003 | 0.9630 | - | - | - | - | - |

**Table 4: Results on MS MARCO V2 document corpus.**

the original corpus seems to yield better results than the augmented corpus. However, we do clearly see the benefits of doc2query–T5 expansions in all cases, independent of the corpus variant.

For passage ranking, the benefits of uniCOIL are shown in rows (3a) and (3b). Here, we performed inference on the original passages. Consistent with all the results we have observed so far on the V1 collections, both learned weights and document expansions contribute to effectiveness. Also consistent with the V1 passage results, uniCOIL with doc2query–T5 expansions beats BM25+RM3 on doc2query–T5 expansions.

Rows (4) and (5) round out our results with a hybrid sparse–dense combination and the best runs from TREC 2021. For dense retrieval, we used the same model as for the V1 results (trained on V1 passage data). We report the best results from TREC 2021 in two categories: the best overall and the best single-stage method.[11] The latter is directly comparable to our experimental conditions, while the former takes advantage of reranking.

[11]Based on metadata collected in 2021 that were not available in 2019 and 2020.

**Document Ranking.** For the document ranking task, we similarly ran baseline bag-of-words techniques (BM25 and BM25+RM3) on the two "flavors" of the document corpus (original and segmented). The preparation of the two different conditions is similar to the V1 data; in both cases, all fields are concatenated together. These results are shown in Table 4, rows (1a) to (1d). Similarly, doc2query–T5 expansions can start from either the original or segmented corpus; these results are shown in rows (2a) to (2d). Comparing the two corpus "flavors", the results seem much more unequivocal: the segmented version consistently achieves higher effectiveness.

Results of the uniCOIL models (only applicable to the segmented corpus) are shown in rows (3a) and (3b). Here, we performed inference on each segment, prepended with the document title. As with previous results, we observe gains of uniCOIL (noexp), row (3a), over BM25 on the segmented corpus, row (1b), once again affirming the value of learning term weights from supervised training data. Document expansions, row (3b), further improve effectiveness. However, the overall benefits of uniCOIL appear more modest. We see that on TREC data, BM25 with doc2query–T5, row (2b), appears to be competitive with the full uniCOIL model, row (3b). Adding

in RM3, row (2d), we actually observe higher average precision than uniCOIL. In fact, even without doc2query–T5, BM25+RM3 on the segmented documents, row (1d), achieves a higher AP than uniCOIL, although nDCG@10 is quite a bit lower. These results suggest that query expansion remains a strong baseline.

Finally, rows (4) and (5) show our sparse–dense hybrid results and the best runs from TREC. The best single-stage run in TREC 2021 originated from our research group, so it's no surprise that our sparse–dense hybrid achieves comparable results. Interestingly, the gap between the best overall run and the best single-stage run is relatively small, which shows the limited benefits of reranking for documents. This is consistent with the above results, which suggest that document ranking is a more difficult task, at least from the perspective of modern neural models.

## 6 CONCLUSIONS

The doc2query method was introduced in 2019, which might as well have been the stone age in the context of rapid progress in the "neural era". However, doc2query–T5 seems to have "aged well" and today remains an important starting point for ranking models based on learned sparse lexical representations. In information retrieval research, since models are developed at specific points in time and evaluated only on resources available at the time, it is often challenging to obtain a broader view of effectiveness across a wide range of experimental conditions. For doc2query and uniCOIL, this paper represents the first time we have exhaustively performed experiments on all meaningful combinations of corpus × model × topic set. This provides exactly the "big picture" that can guide future research. In making every cell in our numerous results tables easily reproducible, we further provide a valuable foundation that others can easily build on.

## ACKNOWLEDGEMENTS

This research was supported in part by the Canada First Research Excellence Fund, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and the Waterloo–Huawei Joint Innovation Laboratory. Computational resources were provided in part by Compute Ontario and Compute Canada.

## REFERENCES

[1] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. 2001. Vector-Space Ranking with Effective Early Termination. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2001)*. 35–42.

[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3* (2018).

[3] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference Proceedings (TREC 2020)*.

[4] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2019. Overview of the TREC 2019 Deep Learning Track. In *Proceedings of the Twenty-Eighth Text REtrieval Conference Proceedings (TREC 2019)*.

[5] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*. Paris, France, 985–988.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota, 4171–4186.

[7] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The Vocabulary Problem in Human-System Communication. *Commun. ACM* 30, 11 (1987), 964–971.

[8] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 3030–3042.

[9] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online, 6769–6781.

[10] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A Community Library for Natural Language Processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic, 175–184.

[11] Jimmy Lin. 2021. A Proposed Conceptual Framework for a Representational Approach to Information Retrieval. *arXiv:2110.01529* (2021).

[12] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *arXiv:2106.14807* (2021).

[13] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*. 2356–2362.

[14] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Morgan & Claypool Publishers.

[15] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. 163–173.

[16] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*. 1723–1727.

[17] 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:1611.09268v1* (2016).

[18] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv:1901.04085* (2019).

[19] Rodrigo Nogueira and Jimmy Lin. 2019. From doc2query to docTTTTTquery.

[20] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. *arXiv:1904.08375* (2019).

[21] Ronak Pradeep, Xueguang Ma, Xinyu Zhang, Hang Cui, Ruizhou Xu, Rodrigo Nogueira, and Jimmy Lin. 2020. $H_2$oloo at TREC 2020: When all you got is a hammer... Deep Learning, Health Misinformation, and Precision Medicine. In *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*.

[22] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. *arXiv:2101.05667* (2021).

[23] Yixuan Qiao, Hao Chen, Liyu Cao, Liping Chen, Pengyong Li, Jun Wang, Peng Gao, Yuan Ni, and Guotong Xie. 2020. PASH at TREC 2020 Deep Learning Track: Dense Matching for Nested Ranking. In *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*.

[24] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[25] Ellen M. Voorhees, Nick Craswell, and Jimmy Lin. 2022. Too Many Relevants: Whither Cranfield Test Collections?. In *Proceedings of the 45th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2022)*.

[26] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online, 38–45.

[27] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor

Negative Contrastive Learning for Dense Text Retrieval. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*.

[28] Ming Yan, Chenliang Li, Chen Wu, Bin Bi, Wei Wang, Jiangnan Xia, and Luo Si. 2019. IDST at TREC 2019 Deep Learning Track: Deep Cascade Ranking with Generation-based Document Expansion and Pre-trained Language Modeling.. In *Proceedings of the Twenty-Eighth Text REtrieval Conference Proceedings (TREC 2019)*.

[29] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017)*. Tokyo, Japan, 1253–1256.

[30] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *Journal of Data and Information Quality* 10, 4 (2018), Article 16.

[31] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. 2019. Critically Examining the "Neural Hype": Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*. Paris, France, 1129–1132.