University of Glasgow Terrier Team at the TREC 2021 Deep Learning Track

Xiao Wang
University of Glasgow, UK
x.wang.8@research.gla.ac.uk

Sean MacAvaney, Craig Macdonald, Iadh Ounis
University of Glasgow, UK
firstname.lastname@glasgow.ac.uk

ABSTRACT

This paper describes our submission to the document ranking and passage ranking tasks of the TREC 2021 Deep Learning Track. In our participation, we conducted the dense retrieval and the sparse retrieval as well as the hybrid of dense and sparse retrieval on both passage ranking and document ranking tasks. For the dense retrieval experiments, we employed the multiple representation ColBERT dense retrieval with and without the pseudo-relevance feedback mechanism implemented. For the sparse retrieval experiments, we experimented with the sparse retrieval model, namely DPH, with and without the query expansion applied, then followed with different types of neural rerankers. For the passage ranking task, we submitted three group runs with the dense retrieval applied: `uogTrPC`, `uogTrPCP` and `uogTrPot6`, and two baseline runs on the inverted index: `uogTrPD` and `uogTrPPD`. For the document ranking task, we submitted three group runs with dense retrieval applied: `uogTrDCPpmp`, `uogTrDot5pmp` and `uogTrDDQ5`, and five sparse retrieval baseline runs. For both tasks, after correction, the hybrid of sparse and dense retrieval runs, namely `uogTrPot5-c` run for passage ranking task and `uogTrDot5pmp-c` run for document ranking task, are most effective.

1 INTRODUCTION

The University of Glasgow Terrier team participated in the TREC 2021 Deep Learning track, in order to improve the effectiveness and flexibility of our new PyTerrier [6, 7] information retrieval (IR) toolkit for an adhoc ranking task on a large corpus of Web documents. PyTerrier leverages Python to allow the easy expression of complex retrieval pipelines. In particular, we focused on the following aspects: (a) Expressing complex ranking pipelines using PyTerrier operators and (b) Testing ColBERT [3] dense retrieval and the ColBERT-PRF [11] approach on the large MSMARCO v2 corpora. In our participation, we addressed both the passage ranking task and the document ranking task of the Deep Learning Track, but without using the provided initial rankings of both tasks.

We conducted the dense retrieval and the sparse retrieval as well as the hybrid of dense and sparse retrieval on both passage ranking and document ranking tasks. For the dense retrieval, we employed the multiple representation ColBERT dense retrieval with and without the pseudo-relevance feedback mechanism implemented. For the sparse retrieval, we used the DPH weighting model, with and without Bo1 query expansion applied, and followed with different types of neural rerankers.

The structure of the remainder of this paper is structured as follows: Section 2 introduces the notions of retrieval pipelines in PyTerrier. Section 3 describes our contribution of TREC DL 2021 data to the `ir-datasets` package and pre-built MS MARCO v2 indices to the public data.terrier.org repository. Section 4 describes the models and the framework we used as well as the experimental setup in this work. The baseline runs and the submitted runs are detailed in Section 5 and Section 6, where Section 5.4 and Section 6.4 highlight our results for each task, respectively. Concluding remarks follow in Section 7.

2 PYTERRIER RETRIEVAL PIPELINES

All of our experiments and submitted runs for the TREC 2021 Deep Learning track are built upon PyTerrier, the expressive Python bindings for Terrier [6]. In particular, PyTerrier defines all retrieval components (rankers or rerankers) to take the form of *transformer* objects, which transform one dataframe to another. To create flexible *pipelines* composed of multiple transformers, PyTerrier overloads standard Python operators for transformer objects as follows:

- `>>` (then): Pass the output of one transformer into another.
- `|` (result set union): Combine the results of two transformers by selecting query-document pairs that appear in either set.
- `%` (Rank cutoff): The `%` operator is called rank cutoff, and limits the number of results returned for each query.

All ranking features described in the rest of this paper were expressed as pipelines of transformers using these operators. We refer the reader to [6] for more information about the PyTerrier platform and the flexibility of the operators.

3 DATA

To improve the ease of working with the TREC DL 2021 datasets, we added them to the `ir-datasets` package [4] (the base dataset IDs are `msmarco-passage-v2` and `msmarco-document-v2`). The software downloads and verifies the dataset's source files (e.g., corpus, query, top100, etc.) as they are needed and provides them in common interoperable formats. We also released sparse Terrier indices for both corpora on data.terrier.org, allowing the datasets to be easily used in experiments on the PyTerrier [6] platform.

We tested two approaches for fast lookups by doc ID. The first uses the conventional lookup structure used in `ir-datasets`, which is a compressed file of document contents and a sorted list of doc IDs with pointers to the location of the document in the compressed file. The second makes use of the source file offset information present in the doc IDs themselves (they indicate the file number and byte offset in the uncompressed source file). We conducted simple benchmarks for lookups based on the TREC DL 2021 reranking documents, loaded from a HDD. For the document corpus, we found that the lookup structure was preferable to source lookups in terms of cold reads (0.75s/query vs. 0.96s/query), cached reads (4ms/query vs. 10ms/query), and storage requirements (62GB vs. 113GB). The speed benefits are largely due to the compression, since less data needs to be read from disk. For the passage corpus, we found the

opposite; the source lookups were preferable to the lookup structure in terms of cold reads (0.6s/query vs. 3.47s/query) and cached reads (1ms/query vs. 4ms/query), with negligible storage overhead (56GB vs. 55GB). Based on these observations, `ir-datasets` uses the lookup structure for the document corpus and source lookups by ID for the passage corpus.

4 METHODS

In this section, we introduce the background knowledge of the multiple representation ColBERT and the ColBERT-PRF dense retrieval models in Section 4.1 and Section 4.2, respectively. This is followed by the introduction of the sparse and dense retrieval hybrid paradigm in Section 4.3, and experimental setup details in Section 4.4.

4.1 ColBERT End-to-End Dense Retrieval

There are two families of dense retrieval models [8]: the single representation dense retrieval model (for instance, the ANCE [12] model), and the multiple representation dense retrieval model (for instance, the ColBERT [3] model). In our participation, we focused on testing the ColBERT dense retrieval performance on large MS-MARCO v2 datasets, as we have found that techniques such as query pruning [10] and PRF [11] can be implemented upon its multiple embedded representations.

In the ColBERT model, a query q is encoded into a set of $|q|$ query embeddings $\{\phi_{q_1}, \ldots, \phi_{q_{|q|}}\}$ and a document is encoded into a set of $|d|$ document embeddings $\{\phi_{d_1}, \ldots, \phi_{d_{|d|}}\}$. For the end-to-end ColBERT dense retrieval model, the document embeddings are pre-computed and can be encoded into a FAISS [2] index, which allows the efficient approximate nearest neighbour search using the query embeddings over the document embeddings. Typically, there are two stages consisting of a ColBERT end-to-end model. In the first stage, a set of k' document embeddings relevant to the query embeddings is identified using the approximate nearest neighbour search, then the returned document embeddings are mapped to a list of k candidate documents. The default setting of $k' = 1000$ results in $k \sim 7000$ documents being retrieved.

Next, in the second exact scoring stage, the contextualised similarity score $s(q, d)$ between a query q and a document d is obtained by summing the maximum similarity score between the query token embeddings and the document token embeddings [3], as follows:

$$s(q,d) = \sum_{i=1}^{|q|} \max_{j=1,\ldots,|d|} \phi_{q_i}^T \phi_{d_j} \tag{1}$$

However, the the computational complexity in the exact scoring stage is proportional to k, the number of candidate documents produced by the first approximate nearest neighbour (ANN) retrieval stage, which can be large. Thus, we adopt the approximate nearest neighbour selection mechanism [5] in the first ANN retrieval stage, which makes use of the scores in the ANN search stage and further produces a smaller candidate ranking of documents to be passed to the exact scoring stage. In particular, we employed the MaxSim method to instantiate a ranking of the candidate documents, which scores and ranks the returned candidate documents by taking for each query embedding the maximum approximate similarity score appearing for any of the document embeddings in the document. Experiments in [5] shows that a candidate set of 300 documents is sufficient to ensure high NDCG@10 effectiveness on v1 of MS MARCO.

More formally, following our notation introduced in Section 2, given some query formulation q, let $Ret(\mathcal{I}, k)(q) \rightarrow R$ denote a retrieval process that takes as input the query q, and returns a ranking of k documents R, obtained from index $\mathcal{I}$. The experimental pipeline for ColBERT end-to-end search can be described as follow:

$$Ret_{ColBERT}(\mathcal{I}, k) = Ret_{ANN_{MaxSim}\ ColBERT}(\mathcal{I}, k) \gg MaxSim(\mathcal{I}, k) \tag{2}$$

where $Ret_{ANN_{MaxSim}\ ColBERT}(\mathcal{I}, k)$ denotes the approximate nearest neighbour selection using MaxSim method and $MaxSim(\mathcal{I}, k)$ denotes the exact maximum similarity reranking stage.

4.2 Dense PRF

In addition, in our participation, we also employed the ColBERT-PRF [11] model. ColBERT-PRF model is built upon the ColBERT model and implemented with PRF mechanism. More specifically, the stages of ColBERT-PRF [11] can be summarised as follows:

(a) After obtaining the top ranked documents and their corresponding document embeddings from the first stage with ANN retrieval, ColBERT-PRF employs KMeans clustering to obtain the representive (centroid) emebddings.

(b) Among the representative embeddings, to identify the most discriminative embeddings, ColBERT-PRF resort to FAISS to obtain the nearest tokens as the most likely tokens for a given representative embedding. By doing so, the Inverse Document Frequency (IDF) of the most likely token is calculated as the weight of the given centroid embedding. This, the representative embeddings with high discriminative power would be selected as the expansion embeddings to be appended to the original query representation.

(c) Finally, to control the emphasis of the expansion embeddings in the final exact scoring stage, a hyperparameter β is used. Thus the weight of an expansion embedding is influenced by both the IDF and β.

Following our notation from earlier, let $PRF_{ColBERT}(\mathcal{I}, \theta)$ denote an instantiation of ColBERT-PRF with parameter θ that performs query embedding expansion on the retrieved documents from an earlier stage. Therefore, the full ColBERT-PRF process applied to a document ranking from ColBERT dense retrieval can be formulated as:

$$\begin{aligned} Ret_{ColBERT-PRF}(\mathcal{I}, k) &= Ret_{ANN_{MaxSim}\ ColBERT}(\mathcal{I}, k) \\ &\gg PRF_{ColBERT}(\mathcal{I}, \theta) \\ &\gg MaxSim_{ColBERT}(\mathcal{I}, k) \end{aligned} \tag{3}$$

4.3 Hybrid of Sparse and Dense Retrieval

Apart from the dense retrieval models, we also implemented the hybrid of sparse and dense retrieval model, whose architecture is shown as Figure 1. In the hybrid of sparse and dense retrieval pipeline, we made combination of sparse retrieval, namely the sparse retrieval using DPH with Bo1 [1] query expansion on the inverted index and ColBERT-PRF dense retrieval on the ColBERT FAISS [2] index, then the obtained documents are further reranked using the monoT5 model on the sparse inverted index.

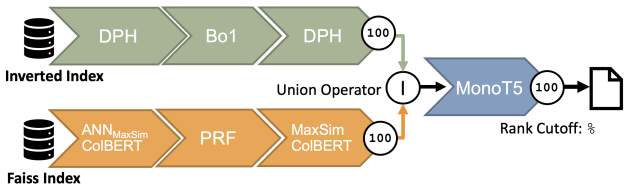

Figure 1: Hybrid of sparse and dense retrieval.

Listing 1: Mapping and MaxPassage Document Runs from Passage Runs.

```
# reading source passage run using PyTerrier
sourcedf = pt.io.read_results(inputfile)
sourcet = pt.transformer.SourceTransformer(sourcedf)
# calculating queries
queries = sourcedf[['qid']].drop_duplicates()
dataset = pt.get_dataset("irds:msmarco-passage-v2")
def make_passaged_docnos(df):
    #actual msmarco doc docnos in "docid"
    df = df.drop(columns={'docid'}, errors='ignore')
    df = df.rename(columns = {'docno' : 'passageno'})
    df["docno"] = df["msmarco_document_id"] + "%p" + df["rank
        "].map(str)
    return df
def docmaxp(input):
    return (input >>
          pt.text.get_text(dataset, "msmarco_document_id",
                by_query=True, verbose=True) >>
          pt.apply.by_query(make_passaged_docnos) >>
          pt.text.max_passage()
    ) % 100
# applying Transformer
res = docmaxp(sourcet)(queries)
# writing the results
pt.io.write_results(res,"/path/to/save/res.gz")
```

Following our notation from earlier, let I_{dense} and I_{sparse} denote the dense index and Terrier sparse index of a given corpora. In creating a hybrid of sparse and dense retrieval systems, we made the union of the returned documents from dense retrieval model, namely ColBERT-PRF, and the returned documents from sparse retrieval model, for instance DPH with or without query expansion. Then the fused documents go through a neural reranker, for instance the monoT5 [9] reranker, and return a final list of reranked documents for further analysis:

$$Ret_{ColBERT-PRF}(\mathcal{I}_{dense}, k) \mid DPH_{w/QE}(\mathcal{I}_{sparse}, k) \gg MonoT5(\mathcal{I}_{sparse}, k) \quad (4)$$

where $Ret_{ColBERT-PRF}(\mathcal{I}_{dense}, k)$ can be obtained using Equation (3).

4.4 Experimental Setup Details

All our experiments were conducted on the PyTerrier [6] IR experimentation platform. PyTerrier is available from https://github.com/terrier-org/pyterrier.

To support dense retrieval: we built the ColBERT dense indices for the `msmarco-passage-v2` following the settings of the original ColBERT [3] paper. However, as `msmarco-passage-v2` is markedly larger than the original `msmarco-passage` corpus, we faced particular technique challenges. Indeed, the ColBERT index for `msmarco-passage-v2` is 2.7 TB in size. Thus, to avoid loading the entire index into memory, we convert the conventional PyTorch format index into a Numpy format index that can be *memory-mapped*, thus facilitating experimentation without needing a machine with 3 TB of RAM. On the other hand, as passage embeddings must be fetched from disk, efficiency was not high, and we were not able to conduct as many experiments as we would have liked.

For ColBERT-PRF model, we employed the ColBERT model checkpoint and followed the default setting of the implementation of the ColBERT-PRF model [11]. The implementation details are available from PyTerrier ColBERT plugin https://github.com/terrierteam/pyterrier_colbert. For the passage ranking dense retrieval runs, we directly operated on the `msmarco-passage-v2` dense index.

For the document ranking task, we did not create a ColBERT index, due to the large size of the `msmarco-document-v2` corpus. Instead, for the document ranking runs with dense retrieval, we obtained the document runs by mapping the `msmarco-passage-v2` passages to the corresponding documents in the document corpus, then applying MaxPassage. Listing 1 provides salient PyTerrier code for performing the mapping and applying MaxPassage.

For the experimental setup of sparse retrieval: we built the Terrier inverted index for `msmarco-passage-v2` and `msmarco-document-v2` corpora, respectively. For the document ranking task, we applied passaging (applying a sliding window of length 150 tokens and stride 75); documents are ranked by applying max passage.

5 RUNS FOR TREC DL 2021 PASSAGE RANKING TRACK

We submitted 3 runs to the passage ranking task. We were also invited by the deep learning track organisers to submit two baseline runs. After submission, we found the implementation of ColBERT-PRF for the submitted runs had a bug, which might caused the performances of the submitted runs to be reduced. Thus, for both passage ranking task and document ranking task, we present additional corrected runs and will analyse the model performances according to the corrected runs.

5.1 Baseline Runs

The baselines that we submitted to the 2021 Deep Learning passage ranking track are constituted of two traditional sparse retrieval runs with and without query expansion, as follows:

- `uogTrPD`: Applying DPH on our passage sparse index.
- `uogTrPPD`: Applying DPH and Bo1 query expansion on our passage sparse index.

5.2 Submitted Group Runs

For the 2021 Deep Learning passage ranking track, we submitted the following 3 runs:

- `uogTrPC`: Conducts ColBERT dense retrieval; the first stage retrieval uses our ANN_{MaxSim} to identify a candidate set of 100 passages.
- `uogTrPCP`: Conducts ColBERT-PRF dense retrieval; the first stage retrieval uses our ANN_{MaxSim} to identify a candidate set of 100 passages.

- uogTrPot6: Conducts a hybrid of the sparse retrieval run, namely uogTrPPD, and the dense retrieval run, namely uogTrPCP, then reranks using the monoT5 model.

5.3 Additional Runs

In addition, we describe three additional corrected dense retrieval runs and two additional sparse retrieval runs.

- uogTrPC-c: Conducts ColBERT dense retrieval; the first stage retrieval uses our ANN_{MaxSim} to identify a candidate set of 100 passages.
- uogTrPCP-c: Conducts ColBERT-PRF dense retrieval; the first stage retrieval uses our ANN_{MaxSim} to identify a candidate set of 100 passages.
- uogTrPot6-c: Conducts the hybrid of the sparse retrieval run, namely uogTrPPD, and the dense retrieval run, namely uogTrPCP, then reranks using the monoT5 model.
- upgTrPPDC: Applies DPH with Bo1 query expansion, then reranks using ColBERT reranker on the passage sparse index.
- upgTrPPDt5: Applies DPH with Bo1 query expansion, then reranks using monoT5 reranker on the passage sparse index.

5.4 Results & Analysis

Table 1 lists the obtained effectiveness results for all our passage ranking task runs, including our baseline runs, submitted group runs and additional runs as well as the TREC per-topic best and median scores across all participating systems, in terms of MAP@100, P@10, NDCG@10 and MRR.

Firstly, we analyse the performance of the baseline runs. Comparing uogTrPD with uogTrPPD, it is clear that sparse query expansion benefited all metrics. Then, we turn our attention to our corrected additional group runs. Among these corrected additional runs, both dense retrieval runs, namely uogTrPC-c run, which performs the ColBERT end-to-end dense retrieval and uogTrPCP-c, which performs the ColBERT-PRF dense retrieval exhibit higher performance than both the sparse baseline runs as well as the TREC Median on all metrics. However, we notice that the reformulated query representation using the additional pseudo-relevance feedback information doesn't bring benefits on the passage v2 corpora. A possible reason behind this is that the default setting of the original ColBERT-PRF is tuned on msmarco-passage-v1 and the TREC 2019 query set, which may not be suitable for the TREC 2021 query set which contains many longer queries than the TREC 2019 query set.

In addition, the uogTrPot5-c run, which conducts the hybrid of the sparse retrieval using DPH with Bo1 query expansion and the dense retrieval using ColBERT-PRF, then reranks using the monoT5 model, achieves the highest performance among all the the group runs. This indicates that the dense retrieval and the sparse retrieval have a complementary effect which can produce the documents with higher quality. Finally, we further compared the sparse retrieval reranked using ColBERT and monoT5 rerankers, namely the uogTrPPDC run and uogTrPPDt5 runs. We find that for passage ranking task, the uogTrPPDt5 outperforms the uogTrPPDC on all metrics.

Analysing the overall trends for our participation in passage ranking task, while it is clear that dense retrieval is overall better than the sparse retrieval on the passage retrieval task, the hybrid of sparse and dense retrieval with the neural reranker applied achieves the best performance. In addition, for passage ranking task, monoT5 is more capable to bring higher quality document to top ranks than ColBERT.

Table 1: Results on TREC Deep Learning track 2021 Passage Ranking track. The best performing run for each measure is emphasised.

	MAP@100	NDCG@10	P@10	MRR
TREC Best (per-topic)	0.4490	0.8369	0.7865	0.9733
TREC Median (per-topic)	0.1941	0.5997	0.5057	0.7680
baseline runs				
uogTrPD	0.1438	0.4619	0.3698	0.3307
uogTrPPD	0.1724	0.4747	0.4000	0.5711
submitted runs				
uogTrPC	0.0860	0.4611	0.3755	0.6119
uogTrPCP	0.0451	0.1389	0.1038	0.2381
uogTrPot5	0.2499	0.6517	0.5736	0.8160
additional runs				
uogTrPC-c	0.2416	0.6264	0.5814	0.7764
uogTrPCP-c	0.2315	0.5958	0.5605	0.7825
uogTrPot5-c	**0.3323**	**0.6904**	**0.6395**	**0.8781**
uogTrPPDt5	0.3117	0.6620	0.5887	0.8274
uogTrPPDC	0.2538	0.6076	0.5302	0.7009

6 RUNS FOR TREC DL 2021 DOCUMENT RANKING TRACK

We submitted 3 runs to the document ranking task. We were also invited by the track organisers to submit baselines runs, which conducted on the sparse index. We chose to submit 5 baseline runs, which also acted as baselines for our group's main submission runs. In addition, we describe 5 additional corrected runs.

6.1 Baseline Runs

Our submitted baseline runs to the document ranking task of the TREC 2021 Deep Learning track for document ranking task conducted on the sparse index, and constituted three runs on the document sparse index and two runs mapping from passage runs. In particular, our 5 baseline runs are as follows:

- uogTrBaseDD: Applies DPH on the document sparse index.
- uogTrBaseDDQ: Applies DPH and Bo1 query expansion on the document sparse index. However, the submitted run was a duplicated submission of uogTrDD, so we report corrected results under uogTrBaseDDQ-c.
- uogTrBaseDDpmp: Applies DPH on our passage sparse index, followed by mapping to documents, and application of MaxPassage.
- uogTrBaseDDQpmp: Applies DPH and Bo1 query expansion on the passage sparse index followed by mapping to documents, and application of MaxPassage.
- uogTrDDQC: Applies DPH with Bo1 query expansion, then reranks using ColBERT reranker on the document sparse index.

6.2 Submitted Group Runs

We submitted the following 3 runs:

- uogTrDCPpmp: Conducts the mapping to documents and application of MaxPassage from the passage run uogTrPC.
- uogTrDot5pmp: Conducts the mapping to documents and application of MaxPassage from the passage run uogTrDot5.
- uogTrDDQt5: Applies DPH with Bo1 query expansion on the document sparse index, then reranked using monoT5 neural reranker.

6.3 Additional Runs

In addition, we describe 4 additional runs, including 3 correction runs and 1 additional run, as follows:

- uogTrBaseDDQ-c: Applies DPH and Bo1 query expansion on the document sparse index.
- uogTrDCpmp: Conducts the mapping to documents and application of MaxPassage from the passage run uogTrPC-c.
- uogTrDCP-c Conducts the mapping to documents and application of MaxPassage from the passage run uogTrPCP-c.
- uogTrDot5pmp-c: Conducts hybrid of the sparse retrieval, namely uogTrBaseDDQ-c, and the dense retrieval run, namely uogTrDCP-c, then reranks using MonoT5 model.

6.4 Results & Analysis

Table 2 lists the obtained effectiveness results for all our document ranking task runs, including our baseline, submitted group runs and additional runs, as well as the TREC per-topic best and median scores across all participating systems, in terms of MAP@100, P@10, NDCG@10 and MRR.

When comparing the performance of the five baselines, firstly, we notice that the runs with query expansion, namely uogTrBaseDDQ-c and uogTrBaseDDQpmp, could improve over the runs without query expansion, namely uogTrBaseDD and uogTrBaseDDpmp on all metrics, respectively. However, the performances of both runs with query expansion, namely uogTrBaseDDQ-c and uogTrBaseDDQpmp, are still lower than the TREC Median metrics. Moreover, when comparing between the two different strategies for obtaining the document runs, i.e. directly retrieve on the document inverted index or conducts mapping to documents and application of MaxPassage, the former strategy results higher performance than the latter strategy. Our explanation for this is that not all passages of the documents are retained in the passage corpus when generating the MSMARCO v2 corpora. Indeed, if the relevant passages are omitted from the passage corpus, a document run only relying on the runs retrieved from the passage corpus would experience degraded recall. Finally, we find that re-ranking using the ColBERT neural reranker on top of the DPH with Bo1 query expansion model, retrieval effectiveness could be further improved. Indeed, as we can see that uogTrBaseDDQC achieves the highest performance among the baseline runs and exhibit higher performance than the TREC Median performance.

Next, we analyse the dense retrieval runs and hybrid of the sparse retrieval and the retrieval runs. As we explained in the previous section, with the error corrected, here we focus on analysing the additional corrected runs. On analysing Table 2, we find that the ColBERT-PRF dense retrieval run, namely the uogTrDCPpmp-c run, exhibits higher MAP@100 score than the ColBERT-E2E model, namely the uogTrDCpmp, but slightly lower than other metrics,

Table 2: Results on TREC Deep Learning track 2021 Document Ranking track. The best performing run for each measure is emphasised.

	MAP@100	NDCG@10	P@10	MRR
TREC Best (per-topic)	0.4384	0.8547	0.9719	0.9868
TREC Median (per-topic)	0.2519	0.6596	0.8298	0.2227
baseline runs				
uogTrBaseDD	0.2487	0.5704	0.7561	0.8297
uogTrBaseDDQ	0.2487	0.5704	0.7561	0.8297
uogTrBaseDDpmp	0.1769	0.5070	0.6772	0.8563
uogTrBaseDDQpmp	0.2029	0.5105	0.6895	0.9649
uogTrBaseDDQC	0.2869	0.6966	0.8456	0.9649
submitted runs				
uogTrDCPpmp	0.0459	0.0115	0.2421	0.3829
uogTrDot5pmp	0.2077	0.6411	0.7842	0.9386
uogTrDDQt5	0.2487	0.7201	0.8596	0.9737
additional runs				
uogTrBaseDDQ-c	0.2894	0.5937	0.7719	0.8599
uogTrDCpmp	0.1869	0.6134	0.7442	0.9128
uogTrDCPpmp-c	0.1938	0.5930	0.7326	0.8969
uogTrDot5pmp-c	**0.3353**	**0.7394**	**0.8581**	**0.9767**

which is consistent with what we have observed in the passage ranking task. Secondly, we present an additional run uogTrDDQt5-c which applies monoT5 reranker on top of the DPH with Bo1 query expansion model. Compared with uogTrBaseDDQC, which conducts DPH with Bo1 query expansion followed by the ColBERT reranker, we find that the uogTrDDQt5 run exhibits marginally higher performance than the uogTrBaseDDQC run in terms of NDCG@10, MRR and P@10 while slightly lower in terms of MAP@100. Finally, similar to the passage ranking task, we also notice that the hybrid of the sparse retrieval and the dense retrieval run, namely the uogTrDot5pmp-c run, achieves the highest performance among all our group runs.

Overall, for the document ranking task, we observe that the dense retrieval based on mapping of passage runs did not outperform the sparse retrieval runs. This can be caused by the information loss for obtaining the document runs by conducting Mapping and application of MaxPassage from passage runs. In addition, we notice that the hybrid of the sparse retrieval and the dense retrieval, followed by a neural reranker run, namely the uogTrDot5pmp-c run, achieves the best performance. The best run in document ranking task is the When comparing our best hybrid run, namely uogTrDot5pmp-c, with the uogTrDDQt5-c run (using the DPH model with query expansion then further reranked using the monoT5 neural reranker model), we can find that uplift of the effectiveness comes from high-quality documents produced by the dense retrieval.

7 CONCLUSIONS

Overall, our participation in the TREC 2021 Deep Learning track was a useful activity to refine methods of integration of deep learning techniques as retrieval pipelines in PyTerrier. We found that: (a) For both the passage ranking task and the document ranking task, in terms of the retrieval effectiveness, the hybrid of dense and sparse retrieval followed with the neural reranker run is the

most effective run among all our group runs as well as the baseline runs; (b) The dense retrieval models for passage ranking task are more effective than all the sparse retrieval models, while for document ranking task, dense retrieval doesn't outperform than the sparse retrieval - this can be caused by the information loss for obtaining the document runs by mapping passages to documents and application of MaxPassage; (c) For sparse retrieval, the monoT5 reranker is more effective than ColBERT reranker when applied on top of DPH with query expansion model. For the future work, we would like to retrain ColBERT model on MSMARCO v2 corpora, though there are some doubts in the community about the quality of the automatically-generated labels in v2. In addition, it would be interesting to test the performance of the single representation dense retrieval on MSMARCO v2 corpora.

REFERENCES

[1] Gianni Amati and Cornelis Joost Van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 357–389.

[2] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[3] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of SIGIR*. 39–48.

[4] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. 2021. Simplified Data Wrangling with ir_datasets. In *Proceedings of SIGIR*.

[5] Craig Macdonald and Nicola Tonellotto. 2021. On approximate nearest neighbour selection for multi-stage dense retrieval. In *Proceedings of CIKM*.

[6] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative experimentation in Python from BM25 to dense retrieval. In *Proceedings of CIKM*.

[7] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2020. Declarative Experimentation in Information Retrieval Using PyTerrier. In *Proceedings of ICTIR*.

[8] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. On Single and Multiple Representations in Dense Passage Retrieval. *IIR 2021 Workshop* (2021).

[9] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713* (2020).

[10] Nicola Tonellotto and Craig Macdonald. 2021. Query Embedding Pruning for Dense Retrieval. In *Proceedings of CIKM*.

[11] Xiao Wang, Craig Macdonald, and Nicola Tonellotto. 2021. Pseudo-Relevance Feedback for Multiple Representation Dense Retrieval. In *Proceedings of ICTIR*.

[12] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *Proceedings of ICLR*.