CHERCHE: A New Tool to Rapidly Implement Pipelines in Information Retrieval

Raphaël Sourty, Jose G. Moreno, Lynda Tamine, François-Paul Servant

CHERCHE: A New Tool to Rapidly Implement Pipelines in Information Retrieval

Raphaël Sourty
raphael.sourty@irit.fr
Université Paul Sabatier, IRIT & Renault
Toulouse, France

Jose G. Moreno
jose.moreno@irit.fr
Université Paul Sabatier, IRIT
Toulouse, France

Lynda Tamine
tamine@irit.fr
Université Paul Sabatier, IRIT
Toulouse, France

Francois-Paul Servant
francois-paul.servant@renault.com
Renault
Boulogne-Billancourt, France

ABSTRACT

In this demo paper, we present a new open-source python module for building information retrieval pipelines with transformers namely CHERCHE. Our aim is to propose an easy to plug tool capable to execute, simple but strong, state-of-the-art information retrieval models. To do so, we have integrated classical models based on lexical matching but also recent models based on semantic matching. Indeed, a large number of models available on public hubs can be now tested on information retrieval tasks with only a few lines. CHERCHE is oriented to newcomers into the neural information retrieval field that want to use transformer-based models in small collections without struggling with heavy tools. The code and documentation of CHERCHE is public available at https://github.com/raphaelsty/cherche

KEYWORDS

Neural Information Retrieval, Python Library, Information Retrieval Pipelines

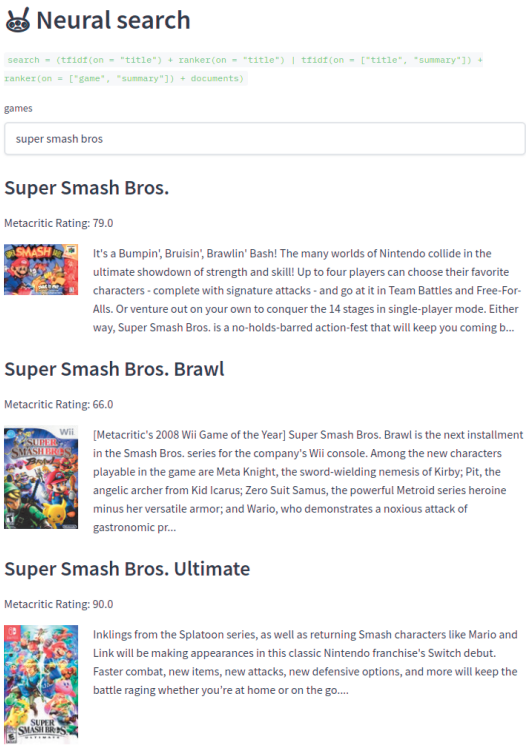

Figure 1: Online demo of CHERCHE using spaces of huggingface available at huggingface.co/spaces/raphaelsty/games.

1 INTRODUCTION

Most recent research in Information Retrieval (IR) is focusing on the integration of neural models within IR, a task which is widely called neural IR. After the development of static embeddings, such as Word2Vec [10], researchers first focused on the use of those static representations into adapted neural models for IR. Most of the models are oriented to propose new architecture that integrate the static embeddings in early layers. Neural tools, such as matchZoo [3], regrouped multiple existing models and facilitated the development of new ones. However, static embeddings are currently being replaced by contextualized representations, such as BERT [1]. These new models have new characteristics that call for the use of the full models instead of a lookup table of embedding vectors [5]. Dedicated libraries to access these new family of models, such as *transformers* [16], were developed and are nowadays widely used in Natural Language Processing (NLP). Some efforts have been made to integrate these libraries in IR systems, but at the expense of extra parameters or new methods on their standard

https://doi.org/10.1145/3477495.3531695

functionalities. Although major advances have been obtained by this new family of models in IR, nowadays there exist few tools capable of perform neural IR based on transformers.

Most of the existing tools for neural IR focus on the training of new models, giving little attention to the use of existing models which makes their integration harder on out of the box IR systems. Indeed, even if an existing pre-trained model for IR is available, its integration on a portable IR system is not a straightforward task. Additionally, it is clear that an important number of IR users may not be interested in training their own models while they are still interested in using recent, and publicly available, advances in the field. Currently, more than 29000 public models are available on the model hub of huggingface, with more than 8000 focused on the use of BERT and more than 300 finetuned for sentence similarity[1].

Our tool, called CHERCHE, was developed as an option to fill this gap. We also aim to empower newcomers in neural IR to explore pipelines with pretrained models without extra effort. Indeed, its flexibility is shown by its integration to online demo portfolios such as spaces of huggingface[2] as shown in Figure 1. Our full architecture is depict in Figure 2. As intended, when using CHERCHE, it is only needed few lines to read, process, and evaluate a neural model as shown in Figure 3[3]. We expect that by reducing the load in the use of these models, more IR users will be motivated to integrate neural IR models into their IR systems.

The main contributions of CHERCHE, and thus, of this demo paper, are:

- a new tool that reduces the load when integrating transformer-based models
- a new option to perform the exploration of new (expert driven) pipelines to improve neural IR models (such as Cascade model [15])

2 RELATED TOOLS

Multiple python-based IR tools are publicly available nowadays. Some of the most popular, pyterrier [8] and pyserini, are backended by their Java versions Terrier and Anserini [17], because, at their time, they are based on Lucene[4]. Both of them are standard, and well established, alternatives when considering a python-based IR tool. Although they both are developed by strong communities, both tools are "heavy"[5] to install and use as extra steps are needed for their use (e.g., starting a Java virtual machine is required in both cases even when only python code is used). This contrasts with the NLP alternatives for similar downstream tasks.

3 CHERCHE

In this section, we detailed CHERCHE. First, we present its installation followed by its raking models, reraker models, pipelines, and finally, its internal evaluation module.

3.1 Installation

We opted for a light installation from pip repository as many other python libraries. So the single line "*pip –install cherche*" allows the full installation of CHERCHE.

3.2 Retriever

Retrievers allow the speed up of a neural search pipeline by filtering out documents that may be not relevant. We implemented most common retrievers based on lexical matching between the query and the documents. However, recent models that use semantic similarity combined with approximate search, based on faiss [4], were also included to speed up the retrieval process. Here is the list of available retrievers using CHERCHE:

- TfIdf: is based on the TfidfVectorizer[6] of sklearn. It computes the dot product between the query tf-idf vector and the documents tf-idf matrix and retrieves the highest match. TfIdf retriever stores a sparse matrix and an index that links the rows of the matrix to document identifiers.
- BM25L and BM25Okapi: is defined by the use of a wrapper to the Rank-BM25[7] library, a two line search engine, which implements the BM25 version of [13].
- Elastic: is defined by the use of a wrapper to the Elasticsearch Python client[8]. This allows the use of any Elasticsearch server into a neural search pipeline. In this case, retrieval parameters are externally defined directly on Elastic.
- Lunr: is defined by the use of a wrapper to Lunr.py[9]. It is a powerful and practical solution for searching inside a corpus of documents. Lunr stores an inverted index in memory.
- Flash: is defined by the use of a wrapper to FlashText[10]. This is based on a keywords-based retrieval system as described in [11].
- Encoder: allows the use of a framework that encodes queries and documents with a single model. It is compatible with the SentenceTransformers models. The encoder pre-computes document embeddings and uses Faiss[11] to quickly find the documents most similar to the query embedding.
- DPR: behaves similarly than Encoder but uses two distinct transformers, one for the document and another for the query.
- Fuzz: is defined by the use of a wrapper to RapidFuzz[12]. It is a blazing fast library dedicated to fuzzy string matching by storing documents in memory.

Currently, only the retriever *Elastic* is recommended with large corpora in CHERCHE. The other retrievers are adapted to small/medium size corpora as they rely on memory to store main data structures.

[1] We arguably suggest that all those models may be useful in neural IR.
[2] https://huggingface.co/spaces/
[3] Full code is available at https://colab.research.google.com/drive/1yN_64KNg6XT6Q5BZ_dzhWlbpnjDOtmfl?usp=sharing
[4] https://lucene.apache.org/
[5] It is clear that current installation process is clean but we refer to heavy to the fact that a Java virtual machine is needed. Thus, library that rely on Java are, arguable, less portable under certain configurations.
[6] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[7] https://github.com/dorianbrown/rank_bm25/
[8] https://github.com/elastic/elasticsearch-py
[9] https://github.com/yeraydiazdiaz/lunr.py
[10] https://github.com/vi3k6i5/flashtext
[11] https://github.com/facebookresearch/faiss
[12] https://github.com/maxbachmann/RapidFuzz

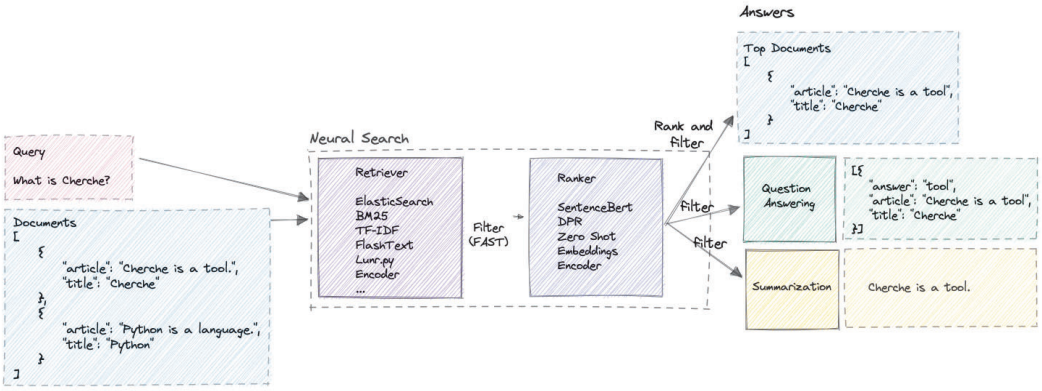

Figure 2: Global architecture of CHERCHE.

```
[ ] 1 !pip install cherche --upgrade
    2 !pip install ir_datasets
    3 !pip install pytrec_eval
```

```
[ ] 1 import pytrec_eval
    2 import pandas as pd
    3 import ir_datasets
    4 from cherche import data, retrieve, rank
    5 from sentence_transformers import SentenceTransformer
```

```
[ ] 1 dataset = ir_datasets.load("vaswani")
```

```
[ ] 1 # List of docs
    2 documents = [{'id':doc,'title':doc,'url':doc,'article':text} for (doc,text) in dataset.docs_iter()]
```

```
[ ] 1 #qrles
    2 qrel = {k: {kk:int(vv) for kk,vv in v[['doc_id','relevance']].values}
    3         for k, v in pd.DataFrame(dataset.qrels_iter()).groupby('query_id')[['doc_id','relevance']]}
```

```
[ ] 1 # Retrieve on field article and evaluate
    2 retriever = retrieve.Lunr(key="id", on=["article"], documents=documents, k=100)
    3 run = {queryid:{x['id']:float(x['similarity']) for i,x in enumerate(retriever(querytext))} \
    4        for queryid, querytext in dataset.queries_iter()}
    5 evaluator = pytrec_eval.RelevanceEvaluator(
    6     qrel, {'map', 'ndcg_cut'})
    7 pd.DataFrame(evaluator.evaluate(run)).T.mean()['ndcg_cut_10']

0.4316086454942083
```

```
[ ] 1 # Rerank on field article
    2 ranker = rank.Encoder(
    3     key = "id",
    4     on = ["article"],
    5     encoder = SentenceTransformer("sentence-transformers/all-mpnet-base-v2",device='cuda').encode,
    6     k = 50,
    7     path = "encoder.pkl"
    8 )
    9
   10 # Pipeline creation
   11 search = retriever + ranker
```

```
[ ] 1 # Preindexing
    2 search.add(documents=documents)
```

```
[ ] 1 # Retrieve and Rerank on field article and evaluate
    2 run = {queryid:{x['id']:float(x['similarity']) for i,x in enumerate(search(querytext))} \
    3        for queryid, querytext in dataset.queries_iter()}
    4 evaluator = pytrec_eval.RelevanceEvaluator(
    5     qrel, {'map', 'ndcg_cut'})
    6 pd.DataFrame(evaluator.evaluate(run)).T.mean()['ndcg_cut_10']

0.4741167462259433
```

Figure 3: Example of running CHERCHE using the vaswani dataset on a fresh Colab notebook. This code include the library installation, dataset reading, qrel preparation, retriever definition, reranking process, and evaluation. Printed values correspond to NDCG@10.

3.3 Reranker

Rerankers will then be able to pull up documents based on semantic similarity. Rerankers are models that measure the semantic similarity between a document and a query. The reranker allows to reorder the documents retrieved by the retriever based on the semantic similarity between the query and the documents retrieved. Rerankers are compatible with all the retrievers in CHERCHE.

To enhance the integration of new models, CHERCHE supports SentenceTransformers models available in the model hub of huggingfaces. This opens a multiple of models that could be used. Additionally, local models can be also specified as the system supports the standard class loader of huggingfaces models.

Here is the list of available rerankers using CHERCHE:

- Encoder: integrates into a neural search pipeline a model capable of building an embedding of a document. This is a cross-encoder strategy.
- DPR: dedicated to the Dense Passage Retrieval models which aims to train two distinct neural networks, one that encodes the query and the other one that encodes the documents.

3.4 Pipelines

CHERCHE overcharges the operators '+' (plus), '|' (union) and '&' (intersection) to build pipelines efficiently.

3.4.1 Operator plus. + is the main operator dedicated to pipelines. This operator allows the definition of a pipeline where left parameter is performed first followed by the second parameter. The number of aggregations is limitless. However, the first model in a pipeline should always be a retriever.

3.4.2 Operator pipe. The union operator '|' is used to improve neural search recall by gathering documents retrieved by multiple models. It can be used when retrieving or reranking. The union will avoid duplicate documents and keep the first one. The first documents out of the union will be from the first model, the next ones will be from the second model and so on. This is not a bug, it is a feature. This strategy allows to prioritize one model or pipeline over another. It may make sense to create a union between two separate pipelines, with the first one having the highest precision and the second one having better recall, like a spare tire.

Table 1: Statistics of the vaswani and scifact datasets.

Dataset	Type	Total	Stats
vaswani	Documents	11429	41.92 tokens/doc
	Qrels	93	22.39 docs/query
scifact	Documents	5183	201.81 tokens/doc
	Qrels	30	1.3 docs/query

3.4.3 Operator intersection. The intersection operator '&' improves the precision of the model by filtering documents on the intersection of proposed candidates of retrievers and/or rankers.

3.5 Evaluation

Evaluate a pipeline using pairs of query and answers. The pipeline objects allow evaluation with three different metrics including F1, Precision, Recall, and P-Recall. However, we used external evaluation libraries in our experiments.

3.6 Example

Figure 3 shows the full python script to perform experiments. Note that, after defining a retriever and a reranker, only one line is needed to define the pipeline, e.g. "*searcher = retriever + ranker*".

4 EXPERIMENTS WITH CHERCHE

In order to highlight the characteristics of CHERCHE, we performed a set of experiments using two datasets of small size and one dataset of medium size. The full code to perform this evaluation is similar to the one in Figure 3, but with a loop over the reranker models to check the performance of other alternatives. In both cases, a simple pipeline strategy was used, e.g. a pipeline composed by a retriever followed by a reranker, as described in Section 3.6.

4.1 Experimental setup

4.1.1 Small size datasets. We used the vaswani[13] and scifact[14] datasets in our first experiments. In both cases, we opted for the python libraries that offer an easy access to the datasets, BEIR [12] and IR_datasets [7], to highlight the flexibility of opting for CHERCHE.

Details of both dataset are presented in Table 1. Note that both dataset are small as experiments were performed without Elastic[15].

4.1.2 Medium size dataset. We also used CHERCHE on MSMarco passages, a medium size dataset. As CHERCHE python retrievers are not stored on disk[16], we used development, TREC DL 2019 and 2020 queries by only indexing query words to avoid large indices[17]. Details of the dataset and qrels are presented in Table 2.

4.1.3 Metrics. Although CHERCHE proposes an internal evaluation module, we opted for standard IR evaluations metrics including MAP, R@1000, NDCG@5, NDCG@10, and NDCG@20. Pytrec_eval [14] and ir-measures [6] implementations were used as evaluation tools.

[13] http://ir.dcs.gla.ac.uk/resources/test_collections/npl/
[14] https://scifact.apps.allenai.org/
[15] Elastic may be needed for larger datasets.
[16] Future versions may include this option.
[17] This may be solved by using ElasticSearch, but the experiment was intended to be light.

Table 2: Statistics of the MSMarco dataset.

Dataset	Type	Total
MSMarco	Documents	8.8M
	Qrels DL 2019	43
	Qrels DL 2020	54

4.2 Models

We used a set of the most downloaded models on huggingfaces hub for the sentence encoders. The full list of models is presented in Table 3.

Table 3: SentenceTransformers models used in our experiments. Models were selected based on their number of downloads only.

Model	Huggingfaces name
Model A	all-mpnet-base-v2
Model B	all-MiniLM-L6-v2
Model C	paraphrase-albert-small-v2
Model D	paraphrase-MiniLM-L6-v2
Model E	paraphrase-mpnet-base-v2
Model F	paraphrase-multilingual-MiniLM-L12-v2
Model G	bert-base-nli-mean-tokens
Model H	LaBSE

4.3 Results on small size datasets

Table 4 presents the summary of our results on both datasets. Note that we included pyterrier as baseline, and if available, we reported the results from their official repository[18] or performed experiments to obtain its results. In both datasets, we used Lunr as retriever as it performs similarly than pyterrier in terms of ndcg@10 when using

[18] https://github.com/terrier-org/pyterrier

Table 4: Results on the two datasets of small size. AVG(A-H) correspond to the average performance of the models A to H. Best model correspond to the best results, the A model. '*' values were taken from https://github.com/terrier-org/pyterrier/blob/master/examples/notebooks/ltr.ipynb.

		vaswani		scifact	
		MAP	NDCG@10	MAP	NDCG@10
pyterrier	PL2	0.2060*	0.4245	0.4087	0.4438
	LambdaMART	0.2043*	-	-	-
CHERCHE	retriever only	**0.2590**	0.4316	0.5169	0.5619
	AVG (A:H)	0.1827	0.3456	0.4494	0.4943
	BEST	0.2508	**0.4741**	**0.6047**	**0.6484**

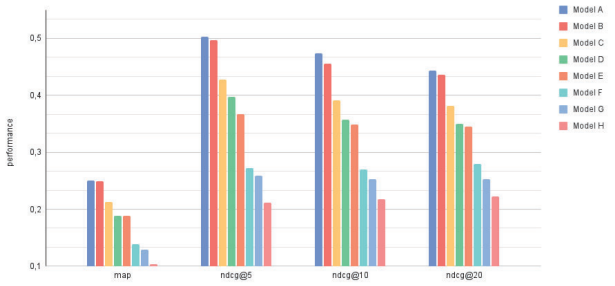

Figure 4: map, ndcg@5, ndcg@10, and ndcg@20 performances for models A, B, C, D, E, F, and H using vaswani dataset and a simple pipeline with CHERCHE.

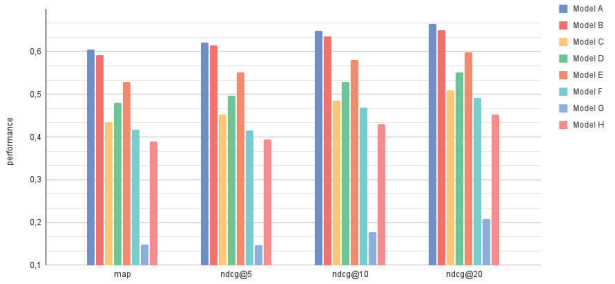

Figure 5: map, ndcg@5, ndcg@10, and ndcg@20 performances for models A, B, C, D, E, F, and H using scifact dataset and a simple pipeline with CHERCHE.

vaswani dataset[19]. As a main result, note that based on Table 4, the best reranker strongly outperforms the retriever, but it is not the case when averaging the performance of all models (AVG). This result highlight the importance of selecting an adapted transformer model, which can be easily performed when using CHERCHE.

4.3.1 vaswani. Results of the performances for the eighth rerankers mentioned in Table 3 are presented in Figure 4. Note that the model A outperforms other alternatives and clearly outperforms the single retriever. Indeed, models A, B, and C outperform the retriever performances. Other models underperformed the retriever. This trend was observed for most of the used metrics.

4.3.2 scifact. Results using scifact are presented in Figure 5 and follow the vaswani results, e.g., the retriever performance is outperformed by a clear margin for models A and B. However, model C did not manage to obtain a good performance, but model E does across multiple metrics. Finally, model G is clearly not an option for this dataset. This shows one of the feature of CHERCHE, the rapidly identification of candidate models to be integrated in a robust pipeline.

[19] Also note that as mentioned before, CHERCHE is clearly an option for small datasets but standards libraries, such as pyterrier, will be more adapted for large datasets.

Table 5: Results on the TREC DL2019 and DL2020 qrels. Best value are in bold and second is underlined. Baseline values were taken from [2, 9].

	TREC DL2019		TREC DL2020	
	NDCG@10	R@1000	NDCG@10	R@1000
retriever only	0.311	0.674	0.351	0.679
Model A	0.687	0.724	**0.684**	0.713
Model B	0.654	0.725	0.658	0.713
Model C	0.668	0.721	0.616	0.708
Model D	0.616	0.720	0.566	**0.715**
Model E	0.672	0.725	0.622	0.710
Model F	0.601	0.709	0.512	0.698
Model G	0.099	0.553	0.104	0.501
Model H	0.249	0.617	0.219	0.623
DeepCT [9]	0.578	-	0.55	-
Siamese [2]	0.637	0.711	-	-
ANCE [2]	0.642	0.827	-	-
DocT5Query [9]	0.648	-	0.619	-
SPLADE [2]	0.665	0.813	-	-
SPLADE-max [2]	0.667	0.747	-	-
SPLADE-doc [2]	0.684	0.851	-	-
DeepImpact [9]	0.695	-	0.651	-
DistSPLADE-max [2]	**0.729**	**0.865**	-	-

4.4 Results on MSMarco

Results of CHERCHE and recent baselines using TREC DL2019 and DL2020 are presented in Table 5. We used the TfIdf retriever for this dataset. As for the other datasets, model A remains a competitive option closely followed by models E and C. Note that none of the models outperform recent contributions such as DistilSPLADE-max [2] or DeepImpact [9] when using the TREC DL2019 qrels. However, Model A obtains the best performance when using the DL2020 qrels but excluding best performing models for DL2019[20].

5 CONCLUSION

This demo paper presents CHERCHE a library for neural pipelines definition. Our library was developed to be light and portable to new environments as a tool to quickly evaluate/integrate neural IR models. Although it can be used to develop new models, CHERCHE targets newcomers to the neural search that want to verify the pipelines based on transformers within their collections. Our example using CHERCHE shows that a full pipeline composed by a retriever followed by a reranker is possible to implement in only a few lines without affecting the retrieval performances.

REFERENCES

[1] 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)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://aclanthology.org/N19-1423

[20] Values were taken from [2] where performances for DL2020 were not reported.

[2] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv preprint arXiv:2109.10086* (2021).

[3] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. 2019. MatchZoo: A Learning, Practicing, and Developing System for Neural Text Matching. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Paris, France) *(SIGIR'19)*. ACM, New York, NY, USA, 1297–1300. https://doi.org/10.1145/3331184.3331403

[4] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[5] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. Pretrained transformers for text ranking: Bert and beyond. *Synthesis Lectures on Human Language Technologies* 14, 4 (2021), 1–325.

[6] Sean MacAvaney, Craig Macdonald, and Iadh Ounis. 2022. Streamlining Evaluation with ir-measures. In *Advances in Information Retrieval*. 305–310.

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

[8] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation inInformation Retrieval using PyTerrier. In *Proceedings of ICTIR 2020*.

[9] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, New York, NY, USA, 1723–1727. https://doi.org/10.1145/3404835.3463030

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[11] Vikash Singh. 2017. Replace or retrieve keywords in documents at scale. *arXiv preprint arXiv:1711.00046* (2017).

[12] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[13] Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to BM25 and Language Models Examined. In *Proceedings of the 2014 Australasian Document Computing Symposium* (Melbourne, VIC, Australia) *(ADCS '14)*. Association for Computing Machinery, 58–65.

[14] Christophe Van Gysel and Maarten de Rijke. 2018. Pytrec_eval: An Extremely Fast Python Interface to trec_eval. In *SIGIR*. ACM.

[15] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 105–114.

[16] 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*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[17] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. 1253–1256.