A Python Interface to PISA!

Sean MacAvaney
University of Glasgow
United Kingdom
sean.macavaney@glasgow.ac.uk

Craig Macdonald
University of Glasgow
United Kingdom
craig.macdonald@glasgow.ac.uk

ABSTRACT

PISA (Performant Indexes and Search for Academia) provides very efficient implementations of various retrieval algorithms over sparse inverted indices. The highly-optimized C++ implementation, however, has previously only been accessible via command line tools. From indexing to retrieval, 5–6 commands need to be executed in sequence, making the process relatively involved. Further complications when using PISA include a lengthy build process and minimal interoperability with other tools. In this work, we demonstrate a new tool that provides a native Python wrapper around PISA. The wrapper features a simplified interface that adheres to the PyTerrier API, making it easy to use (e.g., via Pandas DataFrames), apply to a multitude of datasets (e.g., those from the `ir_datasets` package) and combine with other methods (e.g., neural re-ranking and dense retrieval methods).

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

Retrieval, Inverted Index, Python, Performance

ACM Reference Format:
Sean MacAvaney and Craig Macdonald. 2022. A Python Interface to PISA!. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477495.3531656

1 INTRODUCTION

The humble inverted index – which maps terms to the documents in which they appear – is foundational to information retrieval. Even to this day, sparse inverted indices remain appealing structures, given their simplicity and efficiency. Although alternative structures that facilitate approximate-nearest-neighbour searches over learned dense vectors have the potential to supplant the sparse inverted index, new techniques designed to overcome limitations of inverted indices continue to be developed. For instance, approaches like doc2query [19] overcomes lexical mismatches between queries and documents, while DeepImpact [16] predicts the importance of individual terms within a document better than term frequency.

In this demonstration, we present a Python interface to the PISA engine [18]. PISA provides extremely efficient inverted index algorithms, written in C++. Despite PISA's high efficiency, other (slower) inverted index tools are more commonly used in IR for experiments. At best, this means that researchers waste time/compute on operations that can be done more efficiently. At worst, it can give the incorrect impression that the speed of recent methods (e.g., dense retrieval) are almost at the level of those that use inverted indexes. We posit that one of the main reasons the PISA engine is not more utilised is that it can be challenging to get started using it – builds take a long time, and the existing Command-Line Interface (CLI) is complicated. While many of these details are necessary for those specifically researching the algorithms over inverted indexes, they are not important to many researchers who just want to use the highly-optimized implementations.

This work is the latest in a trend of making tools more convenient by providing interfaces to them from the Python language (e.g., [6, 15]). Among the primary motivations for this are inseparability with various other tools (especially those for deep learning), the popularity of using Jupyter and Google Colab notebooks for interactively working with data, and the general ease of use of the Python language. Compared to languages like Java, interfacing between Python and code written in other languages (e.g., C and C++) is also more straightforward. Infrastructure around Python also helps users get started through its package manager (`pip`), which distributes pre-built packages (eliminating build times).

Rather than developing a brand new API for working with PISA in Python, we opt to write an interface that adheres to the PyTerrier API [15]. This has a number of advantages. First, users already familiar with PyTerrier will be able to start using PISA without learning new semantics.[1] It also allows PISA to be combined with a rich and growing library of other tools because PyTerrier components can be pipelined to construct various indexing and retrieval techniques. For instance, one can use a PISA object as the initial stage of a neural re-ranker, the sparse component of a "hybrid" pipeline, or the target of a doc2query [19] indexing pipeline. And finally, it means that PISA can be used with existing infrastructure for data access, experimentation, and evaluation, all within Python.[2][3]

Since users are able to use our interface in Google Colab notebooks, this work will be easily able to be demonstrated in online or hybrid conference formats. The demonstration will consist of live indexing and retrieval examples over standard benchmark datasets. Participants will also be able to see how the tool can be used as a component of indexing and re-ranking pipelines.

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

[1] Through tutorials [9, 14] and university courses, we have found that PyTerrier semantics are easy for newcomers to learn too. [2] Code and demonstration available at: https://github.com/terrierteam/pyterrier_pisa/ [3] We note that a similar effort was initiated by the PISA maintainers, but there has not been any recent progress https://github.com/pisa-engine/pypisa.

2 REQUIREMENTS

Both PISA and PyTerrier are primarily targetted to academic/research environments. However, we aim to reduce the access barrier to PISA, by hiding the different indexing stages necessary to go from a collection of documents to a working search engine. In doing so, we consider three classes of research users:

- A user who wishes to conduct experiments in an efficient manner, but is agnostic to the underlying implementations.
- A user who wishes to conduct experiments and report timings that can be justified as the state-of-the-art in efficient IR.
- A user who wishes to conduct experiments by varying the underlying indexing and retrieval implementations of PISA.

Along with various dense retrieval backends, PyTerrier already supports two existing sparse search and retrieval backends (namely Terrier [11] and Anserini [22] (which is based on Lucene[4]). Of these, Terrier is an academic project, but does provide state-of-the-art retrieval in the open source (e.g., as we see in Section 4.2, its index is highly-compressed). Anserini is an academic wrapper around the industrially-backed Lucene open source project. Lucene includes some state-of-the-art academic approaches, such as Block-MaxWand [5]. However, both Terrier and Lucene are Java-based, and there is some recognition that the most efficient search implementations consist of low-level "bit-twiddling" operations that are best conducted in native C/C++, rather than virtual machine languages such as Java.

We strongly believe that experimentation in IR is best conducted in powerful high-level languages such as Python. The advent of notebooks environments such as Jupyter and Google Colab allow inexperienced users to run experiments. This drives us to consider carefully a "Pythonic" API that allows to access the key concept of PISA. A good API should be sufficiently memorable such that the user does not require to access the documentation often [4]. Furthermore, it should be *declarative* in nature [13], such that iterative or procedural concepts are hidden.

We choose to implement an API for PISA following conventions in PyTerrier, which allows easy transfer of knowledge developed for other retrieval backends available in PyTerrier, while making available the key functionalities of PISA, such as indexing and retrieval configurations.

3 IMPLEMENTATION DETAILS

Our Python interface to PISA makes use of the Python/C API, [5] which enables it to interact with functionality provided by PISA more efficiently than a 'serialize-invoke-parse' workflow that would interface via invoking the existing PISA CLI [6]. Our interface involves two modules: one written in C++ and one written in Python. The C++ extension module is responsible for interfacing between Python objects (as input/output) and the PISA internals. A Python module wraps the C++ module provides the user with simple access to various functionality and integrates it with PyTerrier.

We use GitHub for version control, but specifically making advantages of GitHub Actions for integration testing. In particular, GitHub Actions permit to run tests for a software library on every pushed commit, and can be replicated across configurations (e.g. various Python versions). Both PyTerrier and Terrier use such detailed testing. We include effectiveness testing within such tests, to ensure that changes of results are clearly captured and checked.

In our experience, PISA can be tricky and time-consuming to build. To speed up installation, we provide pre-built binaries for Linux on Python 3.7–3.8. These are built using the GitHub Actions.

4 DEMONSTRATION

In this section, we show how our Python interface to the PISA engine is used. We demonstrate how to install, index, and perform retrieval using our tool. Along the way, we also show the value in using the tool within Python – it can be easily combined with techniques like document re-writing (e.g., doc2query), hybrid retrieval, and neural re-ranking. We also provide benchmarks and validation to confirm that it is on-par with the official command-line interface to PISA.

Getting started with our new interface on supported systems[6] is as easy as installing any other Python package using `pip`:

```
pip install pyterrier-pisa
```

4.1 The `PisaIndex` Class

Nearly all functionality can be accessed directly from the `PisaIndex` class. The constructor simply takes one required argument (the file path to the index). Named parameters allow the user to override other defaults (such as the stemmer, stop word list, encoding algorithm, and number of worker threads). For instance:

```
from pyterrier_pisa import PisaIndex
idx = PisaIndex('my-index', stemmer='krovetz', threads=16)
```

Note that an index can point to a path where an index does not yet exist; it will just need to be built before it can be used for anything else. This process is covered in the following section.

To reduce the burden of getting started, users can also download prebuilt PISA indexes for several common IR datasets from the data.terrier.org service. For example, to get the MS MARCO [1] passage ranking dataset with default parameters:

```
idx = PisaIndex.from_dataset('msmarco_passage')
```

Alongside indexing and retrieval functionality (covered in detail in subsequent sections), an index object exposes some basic information and index statistics about the index:

```
idx.built() # -> True
idx.num_docs() # -> 8841823
idx.num_terms() # -> 1345701
```

4.2 Indexing

`PisaIndex`'s interface makes indexing any collection of documents easy. The only things that is required is an iterable (e.g., a list or a Python generator) of documents, each of which represented as a dictionary. For example:

[4] https://lucene.apache.org/ [5] https://docs.python.org/3/c-api/index.html

[6] We provide built packages for `manylinux2010` and `manylinux_2_12` platforms on Python 3.7 and 3.8, which encompasses most popular Linux distributions, including Debian, Ubuntu, CentOS and others. The built packages also work on Google Colab.

```
idx.index([
  {'docno': '1', 'text': 'Hello PISA'},
  {'docno': '2', 'text': 'foo bar baz'}
])
```

In many situations, however, IR researchers use standard test collections. At the time of writing, PyTerrier provides access to 150 distinct document corpora via the `ir_datasets` package [10]. The package automatically handles the parsing of a variety of markup and collection formatting (e.g. WARC for ClueWeb, SGML for older TREC corpora, JSON for DPR-W100), which enables consistent preprocessing across systems, and for individual implementations to focus on lexical analysis. For example, to index the MS MARCO passage collection:

```
import pyterrier as pt
# Construct a dataset using the ID from ir-datasets.com
dataset = pt.get_dataset('irds:msmarco-passage')
idx.index(dataset.get_corpus_iter()) # index the corpus
```

Further, a `PisaIndex` can be used in a PyTerrier indexing pipeline. Pipelines allow documents to be rewritten (e.g., splitting long texts into passages, performing doc2query expansion [19], or DeepImpact [16]). This example uses the `pyterrier_doc2query` package[7] to enrich documents with expansion terms:

```
index_pipeline = (
  # Get expansion terms
  Doc2Query(out_attr="exp_terms", batch_size=8) >>
  # Combine original text and expansion terms
  pt.apply.text(lambda r: r['text'] + ' ' + r['exp_terms']) >>
  # Pass resulting enriched documents to the index
  index)
index_pipeline.index(dataset.get_corpus_iter())
```

Verification and Benchmarking. To test how our indexing implementation performs, we compare it against the official PISA command line tools, Terrier, and Anserini. We index two corpora: the MS MARCO passage (v1, 8.8M passages, 1.0GB compressed source) corpus [1] and the TREC Robust 2004 corpus (528k documents, 593MB compressed source). All systems were compared on the same (otherwise idle) machine, reading/writing from the same SSD drive, and using at most 8 worker threads. Further, all systems are instructed to keep stop words, retain the forward index, and merge temporary files. In all cases, we source processed text from `ir_datasets` for as fair of a comparison as possible. However, we also format the data in the way we found best favours the particular implementation. Specifically, for PISA (CLI), we export the data as a plain text file; for Anserini we export as 8 (the number of threads) JSON-lines files of similar size; and for our Python PISA interface and PyTerrier, we read from the `ir_datasets` document iterator.

Table 1 presents the results of our benchmarks. We observe a highly competitive indexing time – faster than all the others explored for MS MARCO, and second-best on Robust04. Interestingly, the indexing speed actually exceeds that of the PISA CLI. Through a variety of tracing, we found that this was overhead sustained through the `stdin` piping mechanism it uses for reading input.[8] We also note the large size of the PISA indexes. The size is because PISA needs to keep an uncompressed version of the forward and inverted index files in order to produce compressed versions optimised for various requested retrieval methods. Terrier has by far the smallest indexes, via Elias gamma and unary encoding [3], but the small size comes at the expense of computational cost during indexing and retrieval. Finally, we confirm the correctness of our implementation by testing that the MD5 hashes of the index files from our interface match the ones produced by PISA's CLI.

Table 1: Indexing performance on MS MARCO (passage, v1) and TREC Robust04. Indexing duration was measured on the same machine under identical conditions, with the data provided in a format favourable to each system.

	`msmarco-passage`		`trec-robust04`	
System	Duration	Size	Duration	Size
PISA (Python, ours)	**97.6s**	4.8GB	52.5s	1.9GB
PISA (CLI)	143.1s	4.8GB	83.5s	1.9GB
Terrier (Python)	350.7s	**2.0GB**	236.9s	**0.5GB**
Anserini (CLI)	252.3s	2.5GB	**43.5s**	0.9GB

4.3 Retrieval

`PisaIndex` provides methods to obtain retrieval functions (called transformers in PyTerrier) for the standard weighting models implemented by PISA, namely BM25, Query Likelihood, PL2 and DPH. These retrieval transformers are exposed through methods such as `.bm25()`. Alongside model-specific parameters (if any), these methods optionally accept the number of results per query and the retrieval algorithm to be used.

Indeed, PISA supports a number of retrieval algorithms, including efficient dynamic pruning strategies such as WAND [2], Block-Max-WAND (BMW) [5], and Variable BMW [17]. One intricacy is that once a PISA index has been created, the `bin/create_wand_data` must be invoked to build the additional index structures to record the score upper bounds necessary for the dynamic pruning strategies. Indeed, exact calculation of upper bounds require knowledge of the relevant weighting model and any parameter settings [12]. To provide a simplified user experience, we made the decision to build these data structures in a "just-in-time" fashion, such that they are created only once a retrieval transformer for a weighting model has been requested. This is done transparently to the user, but can result in extra time taken the first time a particular retrieval configuration is used. These structures are cached to disk, however, so subsequent requests for the same retrieval function do not incur this overhead.

In line with the PyTerrier API, users can perform retrieval simply by calling the `.search()` function of a retrieval transformer:

```
bm25 = idx.bm25(k1=1.7, b=0.3)
bm25.search("the leaning tower")
# qid    docno  rank      score             query
#   1  1094340     1  28.299572  the leaning tower
#   1  4634415     2  28.196728  the leaning tower
# ...      ...   ...        ...               ...
```

[7] https://github.com/terrierteam/pyterrier_doc2query

[8] When we modified the PISA source to accept a file path (rather than using `stdin`) we found the indexing speed to be comparable to our approach. We acknowledge that this unexpected behaviour may be platform-dependent; we only tested on a system running Ubuntu 20.04. We notified the PISA maintainers of this limitation.

Table 2: BM25 Retrieval performance on the MS MARCO (passage, v2) dev set and TREC Robust04. Mean Response Time (MRT) was measured on the same machine under identical conditions, with the queries provided in a format favourable to each system.

	msmarco-passage/dev/small			trec-robust04		
System	RR@10	R@1k	MRT	nDCG@20	R@1k	MRT
PISA (Python, ours)	0.185	0.868	5.2	0.421	**0.698**	2.3
PISA (CLI)	0.185	0.868	**3.9**	0.421	**0.698**	**1.9**
Terrier (Python)	**0.188**	**0.870**	42.2	0.420	**0.698**	18.2
Anserini (CLI)	0.180	0.859	20.7	**0.422**	**0.698**	16.3

The transformer produces output as a Pandas Dataframe (represented as a table above), which can easily be inspected by the user and used for a variety of tasks.

One advantage of the interface's adherence to the PyTerrier API is that it can be easily combined with a multitude of other IR approaches to produce retrieval pipelines such as hybrid sparse-dense retrieval (using the + score combination operator) and neural re-ranking (using the » "then" operator). For instance, a system that combines PISA's BM25 results with ANCE's [21][9] dense retrieval results, then re-ranked by monoT5 [20][10] can be expressed as:

```
ance = ANCERetrieval('ance/model/path', 'ance/index/path')
monot5 = MonoT5ReRanker()
pipeline = (bm25 + ance) >> monot5
pipeline.search('the leaning tower')
```

In most cases, IR research is conducted over standard sets of topics. PyTerrier provides access to over 300 distinct topic sets (and associated relevance assessments). A retrieval transformer can be invoked over an entire set of queries to perform batch retrieval, using multi-threading to parallelise the retrieval:

```
dataset = pt.get_dataset('irds:msmarco-passage/dev/small')
bm25(dataset.get_topics())
#     qid    docno  rank    score                          query
# 1048642  1473037     1  20.0485            what is paranoid sc
# 1048642  7794303     2  18.6497            what is paranoid sc
#     ...      ...   ...      ...                            ...
#  968921  1777277   999  14.3411  where did last names originate
#  968921  6785884  1000  14.3329  where did last names originate
```

At an even higher level, researchers often want to measure the overall performance of retrieval models using standard benchmarks and measures. Here, we show how our Python PISA interface can be evaluated using PyTerrier's `Experiment` API, which calculates system performance using the `ir_measures` package [8].

```
pt.Experiment(                          # Output:
  [idx.bm25(), idx.dph(), idx.qld()],   # name     RR@10    R@1000
  dataset.get_topics(),                 # BM25  0.184974  0.867705
  dataset.get_qrels(),                  #  DPH  0.178215  0.860542
  [MRR@10, R@1000]                      #   QL  0.168317  0.854155
)
```

Validation and Benchmarking. We now demonstrate the effectiveness and efficiency of our interface. Using the same three baseline systems as in our benchmarks (PISA CLI, Terrier, and Anserini), we explore the BM25 performance on the MS MARCO passage dev (small) topics (6,980 natural-language queries) and TREC Robust04 title topics (250 keyword queries). To control for a variety of factors, we use the same machine to run all systems, 'warm' (i.e., recently used) indices from the same drive, a single retrieval thread, the same BM25 parameters (k1=0.9, b=0.4, PISA's defaults), the same stopword list (Terrier's default list), and the same number of results per query (1000). These controls are in place to ensure that a fair of a comparison of the systems. Topics are provided to each system in the most favourable format for each one. We report one standard precision-oriented measure, one recall-oriented measure, and the mean response time per query.

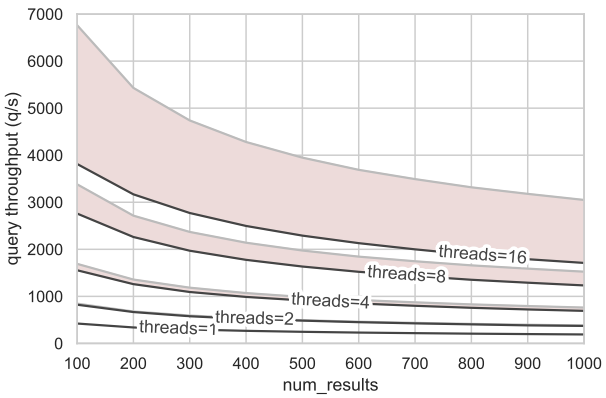

Figure 1: Query throughput of our Python PISA interface by number of results retrieved and number of threads on a 16-core machine. The top of the shaded red areas represent the theoretical maximum performance based on perfect parallelism.

Retrieval results are shown in Table 2. We find that in terms of both precision-oriented and recall-oriented measures, all systems perform comparably. We verify that our interface produces exactly the same results as the official PISA CLI (within the decimal precision of the respective output files). Although our interface incurs 21-33% overhead compared to the PISA CLI in terms of mean response time, our approach still greatly exceeds the performance of Terrier and Anserini (a $4-8\times$ speedup). We plan to look for further ways to optimise our interface in the future, but we suspect an upper-bound exists due to converting data to Python types.

We next examine how well our interface scales to multiple threads during retrieval. We use our interface to retrieve the top 100-1000 documents for each query using BM25 with default parameters, and vary the number of threads used (1, 2, 4, 8, 16). The test is conducted on a machine with 16 cores. We compare the query throughput of each setting compared to the theoretical maximum performance, which would incur no overhead to threading. We show our results in Figure 1. We see that our interface achieves up to 3,813 queries per second (16 threads, 100 results per query). We note that although perfect scaling is not achieved at 16 concurrent threads, it is near-perfect up to 4 threads. Further, we note that our system scales about as well as other systems do a this level.[11]

[9] https://github.com/terrierteam/pyterrier_ance

[10] https://github.com/terrierteam/pyterrier_t5

[11] For reference, Anserini processes 950 queries per second under the same conditions.

Given that our our Python interface to PISA is fast and reasonably scalable, we suspect that it will be a valuable tool for researchers looking to speed up their experiments and compare retrieval performance against a strong baseline.

4.4 Importing and Exporting CIFF

The Common Index File Format (CIFF) [7] is a binary file format used for sharing index structures across search engines. `PisaIndex` supports reading and writing CIFF files using the `from_ciff` and `to_ciff` functions, respectively. This allows indices that were originally built with a different engine to be transferred to PISA's format, and indices that are indexed using PISA to be transferred to other engines. The code sample below demonstrates this process.

```
# importing a CIFF file to PISA:
index = PisaIndex.from_ciff('path/to/index.ciff', 'path/to/index.pisa')
# exporting a CIFF file from PISA:
index.to_ciff('path/to/index.ciff')
```

4.5 Bonus: A Simplified CLI for PISA

Finally, we also provide a CLI to our Python interface, providing value even to those who prefer to work on the command line. For instance, indexing and retrieval can each be conducted using a single command, and the library of datasets are available for experimentation.

```
# Index the MS MARCO passage corpus
pyterrier_pisa index my-index irds:msmarco-passage
# Retrieve from the index using BM25 (results to stdout)
pyterrier_pisa retrieve my-index irds:msmarco-passage/dev bm25
```

We note that although most indexing and retrieval options can be expressed via the CLI, more complicated operations (e.g., pipelines and experiments) are only available through the Python interface.

5 CONCLUSION

In this work we demonstrate a new Python interface to the PISA engine. The interface offers a variety of practical advantages over PISA's existing CLI. Most notably, numerous complicated commands are replaced with just two main operations (index and retrieve). Further indexing and retrieval can easily be augmented with a rich library of other functionality (e.g., document-rewriting and neural re-ranking), and simple access to hundreds of IR benchmark datasets are made available. We expect this library to support a variety researchers that wish to make use of highly-optimised search over sparse indexes. Finally, we hope that this tool will help researchers avoid comparing new approaches against weak baselines in terms of efficiency when reporting the performance of lexical retrieval methods.

ACKNOWLEDGMENTS

We thank Antonio Mallia, Joel Mackenzie, Michał Siedlaczek, and anonymous reviewers for helpful feedback on this manuscript and our implementation. We acknowledge EPSRC grant EP/R018634/1: Closed-Loop Data Science for Complex, Computationally- & Data-Intensive Analytics.

REFERENCES

[1] 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. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proc. of CoCo @ NIPS*.

[2] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*. ACM, 426–434. https://doi.org/10.1145/956863.956944

[3] Matteo Catena, Craig Macdonald, and Iadh Ounis. 2014. On Inverted Index Compression for Search Engine Efficiency. In *Advances in Information Retrieval - 36th European Conference on IR Research, ECIR 2014, Amsterdam, The Netherlands, April 13-16, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8416)*. Springer, 359–371. https://doi.org/10.1007/978-3-319-06028-6_30

[4] Francois Chollet. 2017. User experience design for APIs. https://blog.keras.io/author/francois-chollet.html

[5] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*. ACM, 993–1002. https://doi.org/10.1145/2009916.2010048

[6] Christophe Van Gysel and Maarten de Rijke. 2018. Pytrec_eval: An Extremely Fast Python Interface to trec_eval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*. ACM, 873–876. https://doi.org/10.1145/3209978.3210065

[7] Jimmy Lin, Joel M. Mackenzie, Chris Kamphuis, Craig Macdonald, Antonio Mallia, Michal Siedlaczek, Andrew Trotman, and Arjen P. de Vries. 2020. Supporting Interoperability Between Open-Source Search Engines with the Common Index File Format. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. ACM, 2149–2152. https://doi.org/10.1145/3397271.3401404

[8] Sean MacAvaney, Craig Macdonald, and Iadh Ounis. 2022. Streamlining Evaluation with ir-measures. In *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022 (Lecture Notes in Computer Science)*. https://doi.org/10.1007/978-3-030-99739-7_38

[9] Sean MacAvaney, Craig Macdonald, and Nicola Tonellotto. 2021. IR From Bag-of-words to BERT and Beyond through Practical Experiments. In *Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021 (Lecture Notes in Computer Science)*. Springer.

[10] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. 2021. Simplified Data Wrangling with ir_datasets. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. ACM, 2429–2436. https://doi.org/10.1145/3404835.3463254

[11] Craig Macdonald, Richard McCreadie, Rodrygo L. T. Santos, and Iadh Ounis. 2012. From Puppy to Maturity: Experiences in Developing Terrier. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval, OSIR@SIGIR 2012, Portland, Oregon, USA, 16th August 2012*. University of Otago, Dunedin, New Zealand, 60–63.

[12] Craig Macdonald, Iadh Ounis, and Nicola Tonellotto. 2011. Upper-bound approximations for dynamic pruning. *ACM Trans. Inf. Syst.* 29, 4 (2011), 17:1–17:28. https://doi.org/10.1145/2037661.2037662

[13] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation in Information Retrieval using PyTerrier. In *ICTIR '20: The 2020 ACM SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Norway, September 14-17, 2020*. ACM, 161–168. https://doi.org/10.1145/3409256.3409829

[14] Craig Macdonald, Nicola Tonellotto, and Sean MacAvaney. 2021. IR From Bag-of-words to BERT and Beyond through Practical Experiments. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. ACM, 4861. https://doi.org/10.1145/3459637.3482028

[15] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. ACM, 4526–4533. https://doi.org/10.1145/3459637.3482013

[16] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. ACM, 1723–1727. https://doi.org/10.1145/3404835.3463030

[17] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. ACM, 625–634.

https://doi.org/10.1145/3077136.3080780

[18] Antonio Mallia, Michal Siedlaczek, Joel M. Mackenzie, and Torsten Suel. 2019. PISA: Performant Indexes and Search for Academia. In *Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France, July 25, 2019 (CEUR Workshop Proceedings, Vol. 2409)*. CEUR-WS.org, 50–56. http://ceur-ws.org/Vol-2409/docker08.pdf

[19] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. arXiv:1904.08375 [cs.IR]

[20] 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 [cs.IR]

[21] 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 *Proc. ICLR*.

[22] 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, Shinjuku, Tokyo, Japan, August 7-11, 2017*. ACM, 1253–1256. https://doi.org/10.1145/3077136.3080721