Aligning the Research and Practice of Building Search Applications: Elasticsearch and Pyserini

Josh Devins,[1] Julie Tibshirani,[1] and Jimmy Lin[2]

[1] Elastic NV

[2] David R. Cheriton School of Computer Science, University of Waterloo, Canada

ABSTRACT

We demonstrate, via competitive bag-of-words first-stage retrieval baselines for the MS MARCO document ranking task, seamless replicability and interoperability between Elasticsearch and the Pyserini IR toolkit, which are both built on the open-source Lucene search library. This integration highlights the benefits of recent efforts to promote the use of Lucene in information retrieval research to better align the research and practice of building search applications. Closer alignment between academia and industry is mutually beneficial: Academic researchers gain a smoother path to real-world impact because their contributions can be more easily deployed in production applications. Industry practitioners gain an easy way to benchmark their innovations in a rigorous and vendor-neutral manner by exploiting evaluation resources and infrastructure built by the academic community. This two-way exchange between academia and industry allows both parties to "have their cakes and eat them too".

CCS CONCEPTS

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

KEYWORDS

academia–industry collaborations; open-source software

ACM Reference Format:

Josh Devins, Julie Tibshirani, and Jimmy Lin. 2022. Aligning the Research and Practice of Building Search Applications: Elasticsearch and Pyserini. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3488560.3502186

1 INTRODUCTION

In recent years, the academic community has seen efforts to better align research in information retrieval with the practice of building real-world search applications through greater adoption of the open-source Lucene search library. Today, outside a handful of web search engine companies that deploy custom infrastructure, the Lucene search library and platforms that are built on top of it, such as Elasticsearch and Solr, provide the preferred starting point for real-world production search applications. Examples include large-scale deployments at Bloomberg, Disney, Instagram, Netflix, Spotify, Twitter, Wikipedia, and Yelp.

Although early in its development Lucene was demonstrably inferior in ranking effectiveness [15], recent benchmarks have shown that for bag-of-words ranking, Lucene's effectiveness is at least on par, if not better, than academic search engines [2, 7]. Lucene appears more than capable of filling the role of first-stage retrieval based on inverted indexes in a multi-stage ranking architecture. In terms of query efficiency, while Lucene still lags behind the best research systems such as PISA [11], it nevertheless achieves competitive performance and is faster than some academic search engines. The recent addition of Block-Max Wand [4] has contributed to further increases in performance. These observations raise the question: Why wouldn't academic researchers take greater advantage of Lucene in their work? Adoption does not require compromises to either effectiveness or efficiency, and furthermore grants access to the benefits of a widely deployed, industrial-strength solution with a large ecosystem.

Within this context, our work presents a case study illustrating improved alignment between academia and industry via the adoption of Lucene. We show that competitive bag-of-words first-stage retrieval baselines for the popular MS MARCO document ranking task can be easily reproduced in both the popular Elasticsearch search engine as well as Pyserini [8], the Python interface to the Anserini IR toolkit [16]. The former is a widely used, industrial-strength search platform, while the latter is a research-oriented toolkit that underlies much recent work on ranking models built using pretrained transformers. The interoperability is straightforward because both Elasticsearch and Pyserini are built on the open-source Lucene search library.

Why is this important? This demonstration of seamless interoperability between Elasticsearch and Pyserini means that a two-way exchange between industry and academia is possible in a manner that hasn't been practical until now. For academic researchers, innovations built on top of Pyserini can be migrated to Elasticsearch to enable them to be deployed at scale. This provides a clear path for research to achieve broader impact in industry applications. For practitioners and builders of real-world search applications, there is now an easy path to benchmark search effectiveness in a rigorous and vendor-neutral manner, by leveraging evaluation resources and infrastructure built by academic researchers. Moving forward, we believe this alignment will yield mutually beneficial relationships and create positive feedback loops. In other words, there is no longer a trade-off between rigor in building effective ranking models and the potential for broader impact in real-world search applications. We hope that this case study can lead to further collaborations between academia and industry.

WSDM '22, February 21–25, 2022, Tempe, AZ, USA.

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3502186

2 THE RIGHT TOOL FOR THE JOB

Modern search applications—both real-world production deployments as well as designs primarily for academic study—are dominated by multi-stage ranking architectures, where an initial "first-stage" retrieval system or module produces a list candidates that are then reranked by subsequent modules (this process is also called candidate generation). In some cases, a single reranking model is used, but others have described intricate multi-module reranking pipelines that progressively prune down and refine the list of candidates. Previously, these reranking modules exploited manually-engineered features in a learning to rank framework [6, 10], but more recently, pretrained transformers based, for example, on BERT, have come to dominate the landscape [9]. Within multi-stage ranking architectures, our focus is on first-stage retrieval, typically via bag-of-words ranking using BM25.

What is Lucene? At the core, Lucene is a *search library*. It provides all the components needed for search: document processing pipelines, inverted indexing capabilities, compression codecs, query execution algorithms, as well as implementations of popular scoring functions like BM25. However, it does not prescribe how one would assemble these parts into a search application, or offer a way to index and query the data via convenient interfaces (e.g., a REST API). For this, many developers turn to a fully featured search engine such as Elasticsearch.

When is Elasticsearch used? Elasticsearch was designed as a platform for large-scale, high-performance search over large corpora of structured and unstructured documents. Elasticsearch wraps Lucene and adds the concept of a shard, which corresponds to a single Lucene index. For scalability and performance, users typically create several shards spread over separate physical machines. Elasticsearch manages these distributed shards through the concept of a "cluster". At index time, Elasticsearch distributes documents across the cluster and ensures well-balanced shards. At search time, Elasticsearch sends the query to each shard and collects the top-k results from each one, then merges them together to find the global top-k results. Scoring in Elasticsearch is flexible and can range from Lucene's standard BM25 implementation to custom scripts and ranking functions (more below). To provide reliability and high availability, each shard can be replicated, so that more than one machine contains a copy of the data.

Given that Elasticsearch caters specifically to developers and system builders deploying applications into production environments—who may or may not have a background in information retrieval—the design goals of Elasticsearch have been focused on this audience. As an example, Elasticsearch uses a Query DSL instead of exposing lower-level Lucene primitives. This allows Elasticsearch to broaden its user base and to support use cases beyond the typical text search scenario. For instance, Elasticsearch is commonly used as a datastore for log and telemetry data from applications and servers.

Elasticsearch provides developers with a number of customization options. Through scripting and a Java-based plugin architecture, search application developers can customize first-stage retrieval beyond the standard Lucene query types and scoring functions. It also supports reranking through either scripting or a plugin: One popular example is a learning-to-rank plugin co-developed by Wikimedia and Open Source Connections.[1] In some architectures, Elasticsearch results are fed into downstream components such as QA readers [5] as part of an end-to-end application.

To summarize, Elasticsearch provides a convenient framework for building and deploying production search applications at scale.

When is direct Lucene access preferable? In contrast to the development of large-scale production search applications, academic researchers typically have more focused goals. In systems-oriented research, efforts are frequently focused on improving ranking models with respect to standard benchmark test collections, such as those from TREC and large-scale datasets such as MS MARCO [1]. Frequently, work focuses on rerankers in a multi-stage ranking pipeline, for example, based on pretrained transformers.

In this context, first-stage retrieval is usually fixed and not the focus of innovation. Thus, researchers have different demands that mean Elasticsearch is not necessarily the best solution. At the top of the wish list is usually ease of use and support for rapid iteration with standard IR test collections; that is, simplified access to corpora, queries, evaluation tools, etc. These components are integral to the "research lifecycle" of systems-oriented IR research. Another important feature is reproducibility, since first-stage retrieval is the foundation of complex reranking architectures. The first-stage retrieval must be stable and provide consistent output so that it is possible to compare downstream rerankers in a fair manner.

From this perspective, Elasticsearch offers little support, e.g., parsers and ingesters for standard document formats such as TREC's SGML markup, support for query input and batch run output formats, integration with evaluation tools such as `trec_eval`. While such support could be built into Elasticsearch, a more fundamental issue is that its REST-focused API adds friction for researchers only interested in batch evaluations. For example, querying in Elasticsearch requires issuing REST calls using its DSL, and for most research scenarios where the document collection can comfortably reside on a single machine (the same machine the researcher is querying from), many of the features offered by Elasticsearch aren't necessary—and in fact, add additional overhead. These include, for example, the cognitive overhead of needing to start an Elasticsearch instance prior to any experiments distinct from the "driver" program issuing the queries, or the performance overhead of sending and receiving REST requests from the same machine.

Furthermore, researchers often require access to lower-level primitives that Elasticsearch has chosen not to expose. For example, there is sometimes the need for researchers to manually traverse postings lists and manipulate raw document vectors, which is not straightforward to accomplish with the Elasticsearch API. For these reasons, researchers desire direct access to Lucene.

When should Anserini or Pyserini be used? This is where Anserini comes in: the toolkit provides exactly those features that are missing in the Lucene search library to support the "research lifecycle". These include ingestion support for standard corpora, parsers for different query formats, utilities for orchestrating standard batch runs, etc. In addition, Anserini provides a suite of rigorous regression tests that cover dozens of IR test collections, such that reproducing a standard "bag of words" baseline using, say, BM25 with

[1] https://diff.wikimedia.org/2017/10/17/elasticsearch-learning-to-rank-plugin/

RM3 query expansion, is as simple as copying and pasting a few commands from Anserini's online documentation into a shell.

Given that Lucene is implemented in Java, these Anserini features are implemented in Java as well. However, Python has emerged as the language of choice for researchers today, driven in part by the dominance of toolkits for deep learning such as PyTorch, which has adopted Python as its front-end language. As a response to this demand, we have developed Pyserini [8], which provides Python bindings to Anserini, allowing access to its features from directly within Python. This makes it much easier, for example, to integrate first-stage retrieval with rerankers built on cross-encoder models based on BERT and other transformers.

3 EXPERIMENTAL RESULTS

To illustrate how the research and practice of building search applications can be brought into closer alignment, we present a case study with the MS MARCO document ranking task [1]. Over the past couple of years, the MS MARCO datasets have been instrumental in helping researchers explore *ad hoc* retrieval in the "large-data regime", particularly in the context of data-hungry neural network models [3]. These are among the most widely used resources today for IR as well as NLP tasks, with competitive leaderboards that attract participation from many teams around the world, both from academia as well as industry.

Here, we focus on the MS MARCO document ranking task, which is a standard *ad hoc* retrieval task that uses a corpus of 3.2M web pages. Each document is comprised of three fields (URL, title, and body text) and averages around 1130 tokens of content (median, 580 tokens). The dataset contains a training set with 367K queries and a development set with 5193 queries, with exactly one relevance judgment per query. There are 5793 test queries whose relevance judgments are not publicly available; scores on these test queries can only be obtained by a submission to the leaderboard.

The official metric for the task is mean reciprocal rank at a cutoff of 100 (MRR@100). To evaluate a particular ranking model for first-stage retrieval, it is helpful to measure recall, since that provides the upper bound of effectiveness for any reranking architecture. Thus, recall at a cutoff of 1000 is typically reported on the development set to accompany MRR scores—and arguably, recall is more important than MRR in the context of multi-stage architectures. Since relevance judgments are not publicly available for the test set, recall on the test set is not available.

In Table 1, we present results on the MS MARCO document ranking task, showing the effectiveness of three different sets of runs: those in rows (1a–f) denote runs with Pyserini, those in rows (2a–b) denote runs with Elasticsearch, and row (3a) denotes an Elasticsearch run replicated with Pyserini.

For document retrieval, the obvious baseline is to consider each document as the unit of indexing. These results are shown in rows (1a) and (1b), with default BM25 parameters and after optimizing for recall@100 using grid search on a subset of the dev queries.

An alternative configuration is to first segment each document into multiple passages and consider each as a separate unit of indexing (i.e., treat each passage as a separate "document"). At retrieval time, a document ranking is constructed by retaining only the top-scoring passage from each document. There are a couple of reasons for this design: First, it reduces the amount of text that is passed to downstream rerankers that may be computationally expensive. Second, it provides a fair comparison to dense retrieval techniques, which are today limited in the length of text that they can encode. Rows (1c) and (1d) show the results of this approach with default BM25 parameters and after parameter tuning (same procedure as above), respectively. With default parameters, we observe a large effectiveness gap between (1a) and (1c), the per-document vs. per-passage approaches. However, with tuned parameters, (1b) vs. (1d), both conditions exhibit comparable effectiveness.

		MS MARCO Document		
		Development		Test
Method		MRR@100	R@1k	MRR@100
Anserini/Pyserini				
(1a)	Original text (doc)	0.230	0.886	0.201
	BM25, default ($k_1 = 0.9, b = 0.4$)			
(1b)	Original text (doc)	0.277	0.936	-
	BM25, tuned ($k_1 = 4.46, b = 0.82$)			
(1c)	Original text (passage)	0.268	0.918	-
	BM25, default ($k_1 = 0.9, b = 0.4$)			
(1d)	Original text (passage)	0.275	0.931	0.246
	BM25, tuned ($k_1 = 2.16, b = 0.61$)			
(1e)	doc2query–T5 (doc)	0.327	0.955	0.291
	BM25, tuned ($k_1 = 4.68, b = 0.87$)			
(1f)	doc2query–T5 (passage)	0.321	0.953	0.290
	BM25, tuned ($k_1 = 2.56, b = 0.59$)			
Elasticsearch				
(2a)	Original text (doc)	0.308	-	0.268
	Optimized disjunction max			
(2b)	doc2query–T5 (doc)	0.344	-	0.300
	Optimized boolean			
Elasticsearch/Pyserini Integration				
(3a)	Original text (doc)	0.307	-	-
	Optimized disjunction max			

Table 1: Results on the MS MARCO document ranking task.

Rows (1e) and (1f) show document expansion results using our doc2query method [13], specifically with the T5 model [14] as described in Nogueira and Lin [12]. In both cases, the expansions are appended to the unit of indexing (document or passage). Results show that doc2query increases effectiveness without incurring costs associated with neural inference at query time.

Rows (2a)–(2b) show per-document retrieval with Elasticsearch, using multiple fields and optimizing the score contribution of each field along with other available query parameters. When considering how to index MS MARCO documents in Elasticsearch, one would typically keep fields separate for two reasons: (1) to allow the document score to incorporate matches in different fields separately, and (2) to be able to search on any one of the fields or combinations thereof. A key insight in these experiments is that document structure can provide valuable ranking signals. Instead of collapsing each document's title, body, and URL fields into a single piece of text, we preserve the document structure in the index and take advantage of it in scoring. This is possible thanks to Lucene's design, which allows dedicated configurations and search strategies per document field. Elasticsearch provides built-in support for searching across multiple fields through its Query DSL; contributions from each field can be combined via weights, called 'boosts'.

For MS MARCO, our hypothesis was that using Elasticsearch's support for multi-field scoring would give us an edge over the existing Anserini BM25 baseline. We have three fields: URL, title, and body. The task of selecting appropriate weights can be framed as a black-box optimization problem. With several techniques and open-source libraries available, we chose Bayesian optimization and `skopt` as a well-known approach. Using a sample (1,000 documents) from the MS MARCO training dataset, we were able to choose optimal field boosts.

Row (2a) shows the outcome of this experiment. We used the `multi_match` query type from Elasticsearch 7.10. The run uses a popular option called *best fields*, which scores each field through BM25 and takes the maximum as the overall query score. In row (2b) we added a separate field for doc2query–T5 expansions, bigram fields for each original field, and per-field BM25 parameter tuning. These submissions provide evidence that multi-field retrieval and scoring is useful in first-stage retrieval.

Row (3a) shows the same experiment as in (2a) but performed in Anserini using the field boosts found through the optimization process in (2a). To achieve this, we extended Anserini to support configurable multi-field scoring. As mentioned above, Elasticsearch provides a built-in approach to multi-field scoring that takes the maximum field score as the overall query score. Under the hood, the implementation is powered by Lucene's *disjunction max query*. Since Anserini also delegates core scoring to Lucene, we were able to port Elasticsearch's approach simply by 'plugging in' new Lucene classes and exposing their configurations. Because of the shared foundation, we could also replicate the tokenization strategy from experiment (2a) entirely through Lucene configuration options.

Illustrating this alignment between Elasticsearch and Pyserini, it is possible for researchers to reproduce these experimental runs quite easily. For example, a single command yields run (1b):

```
$ python -m pyserini.search --topics msmarco-doc-dev \
    --index msmarco-doc --output run.msmarco-doc.doc.txt \
    --output-format msmarco --hits 100 --bm25
```

Internally, Pyserini already "knows about" the MS MARCO document dataset: By specifying the `msmarco-doc` index, the toolkit will automatically download (and cache locally) the appropriate pre-built inverted index from a known network location. Similarly, `msmarco-doc-dev` corresponds to an internally known symbol: Pyserini will retrieve (and locally cache) queries and relevance judgments from a known location. We can evaluate the effectiveness of the run with another simple command (not shown) to obtain an MRR@10 score that corresponds to the entry in Table 1.

Similarly, run (3a) can be reproduced using an easy-to-follow guide that we have prepared[2] with the following command:

```
$ python -m pyserini.search --topics msmarco-doc-dev \
    --index indexes/msmarco-doc/lucene-index-msmarco/ \
    --output runs/run.msmarco-doc.leaderboard-dev.elastic.txt \
    --output-format msmarco \
    --hits 100 --bm25 --k1 1.2 --b 0.75 \
    --fields contents=10.0 title=8.63280262513067 url=0.0 \
    --dismax --dismax.tiebreaker 0.3936135232328522 \
    --stopwords docs/elastic-msmarco-stopwords.txt
```

In other words, what can be done in Elasticsearch can be easily done in Pyserini as well!

[2]https://github.com/castorini/pyserini/blob/master/docs/experiments-elastic.md

4 RESEARCH TO PRODUCTION AND BACK

The development of this demonstration illustrates the potential benefits of better aligning the research and practice of build search applications. Chronologically, baseline runs on the MS MARCO document ranking task were first developed in Pyserini/Anserini; these are rows (1a–f) in Table 1. As a way to rigorously benchmark the effectiveness of Elasticsearch and its features, the first author (from Elastic NV) demonstrated how the platform can be easily adapted for the ranking task, and even incorporated a research innovation (document expansion) in a submission; these are rows (2a) and (2b) in Table 1.[3]

In this collaboration, we further showed that Elasticsearch retrieval models can be ported back to Pyserini given their shared reliance on Lucene. This makes the Elasticsearch experiments into a foundation that researchers can subsequently build on. In summary, this case study illustrates innovations from research to a production-ready application and back. With the closer alignment between academia and industry advocated here, we expect such examples to become increasingly common. This yields mutually beneficial and virtuous cycles of engagement, in short, allowing us to "have our cake and eat it too".

[3]https://www.elastic.co/blog/improving-search-relevance-with-data-driven-query-optimization

REFERENCES

[1] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3*.

[2] R. Clancy, N. Ferro, C. Hauff, J. Lin, T. Sakai, and Z. Wu. 2019. Overview of the 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In *Open-Source IR Replicability Challenge at SIGIR 2019*.

[3] Nick Craswell, Bhaskar Mitra, Daniel Campos, Emine Yilmaz, and Jimmy Lin. 2021. MS MARCO: Benchmarking Ranking Models in the Large-Data Regime. In *SIGIR*. 1566–1576.

[4] A. Grand, R. Muir, J. Ferenczi, and J. Lin. 2020. From MaxScore to Block-Max WAND: The Story of How Lucene Significantly Improved Query Evaluation Performance. In *ECIR*.

[5] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*. 6769–6781.

[6] H. Li. 2011. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers.

[7] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *ECIR*.

[8] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *SIGIR*. 2356–2362.

[9] J. Lin, R. Nogueira, and A. Yates. 2021. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Morgan & Claypool Publishers.

[10] T.-Y. Liu. 2009. Learning to Rank for Information Retrieval. *FnTIR* 3, 3 (2009), 225–331.

[11] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. 2019. PISA: Performant Indexes and Search for Academia. In *Open-Source IR Replicability Challenge at SIGIR 2019*.

[12] R. Nogueira and J. Lin. 2019. From doc2query to docTTTTTquery.

[13] R. Nogueira, W. Yang, J. Lin, and K. Cho. 2019. Document Expansion by Query Prediction. *arXiv:1904.08375*.

[14] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR* 21, 140 (2020), 1–67.

[15] H. Turtle, Y. Hegde, and S. Rowe. 2012. Yet Another Comparison of Lucene and Indri performance. In *SIGIR 2012 Workshop on Open Source Information Retrieval*.

[16] P. Yang, H. Fang, and J. Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *JDIQ* 10, 4 (2018).