Anserini: Reproducible Ranking Baselines Using Lucene

PEILIN YANG and HUI FANG, University of Delaware
JIMMY LIN, University of Waterloo

This work tackles the perennial problem of reproducible baselines in information retrieval research, focusing on bag-of-words ranking models. Although academic information retrieval researchers have a long history of building and sharing systems, they are primarily designed to facilitate the publication of research papers. As such, these systems are often incomplete, inflexible, poorly documented, difficult to use, and slow, particularly in the context of modern web-scale collections. Furthermore, the growing complexity of modern software ecosystems and the resource constraints most academic research groups operate under make maintaining open-source systems a constant struggle. However, except for a small number of companies (mostly commercial web search engines) that deploy custom infrastructure, Lucene has become the *de facto* platform in industry for building search applications. Lucene has an active developer base, a large audience of users, and diverse capabilities to work with heterogeneous collections at scale. However, it lacks systematic support for *ad hoc* experimentation using standard test collections. We describe Anserini, an information retrieval toolkit built on Lucene that fills this gap. Our goal is to simplify *ad hoc* experimentation and allow researchers to easily reproduce results with modern bag-of-words ranking models on diverse test collections. With Anserini, we demonstrate that Lucene provides a suitable framework for supporting information retrieval research. Experiments show that our system efficiently indexes large web collections, provides modern ranking models that are on par with research implementations in terms of effectiveness, and supports low-latency query evaluation to facilitate rapid experimentation

CCS Concepts: • **Information systems** → **Retrieval models and ranking**; **Retrieval effectiveness**; **Retrieval efficiency**; **Search engine architectures and scalability**;

Additional Key Words and Phrases: *Ad hoc* retrieval, TREC

ACM Reference format:
Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *J. Data and Information Quality* 10, 4, Article 16 (October 2018), 20 pages.
https://doi.org/10.1145/3239571

16

1 INTRODUCTION

As information retrieval is primarily an empirical discipline, advances are built on experimental validation of proposed methods. Critical to continued progress in better ranking models are

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the U.S. National Science Foundation under IIS-1423002 and CNS-1405688. Any opinions, findings, and conclusions or recommendations expressed do not necessarily reflect the views of the sponsors.
Authors' addresses: P. Yang and H. Fang, University of Delaware, Newark, DE 19716, USA; emails: yangpeilyn@gmail.com, hfang@udel.edu; J. Lin, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada; email: jimmylin@uwaterloo.ca.

1936-1955/2018/10-ART16 $15.00
https://doi.org/10.1145/3239571

baselines that serve as a means for measuring, calibrating, and contextualizing proposed contributions. Baselines appropriate to the subject of study need to be considered, of course, but researchers nearly always include comparisons against a bag-of-words ranking model such as BM25 [47] and query likelihood in the language modeling framework [45].

Unfortunately, researchers often do not pay enough attention to both the description and execution of the baselines. For example, an author might write something like "we used BM25 (or query likelihood)" in the methods section of a paper without further elaboration. Such a statement, of course, is an under-specified description. Frequently missing are the parameter settings, e.g., k_1 and b for BM25, and μ for Dirichlet smoothing. For example, Robertson et al. [47] quite explicitly stated that "BM25 is referred to as BM25(k_1, k_2, k_3, b)." However, most researchers neglect to specify the appropriate parameters in their papers.

To complicate matters, Trotman et al. [52] noted that there are at least half a dozen different variants of scoring functions commonly referred to as BM25 or query likelihood. In some cases, effectiveness differences between these variants are statistically significant. So, which variant is an author referring to in a paper? Similarly, Mühleisen et al. [41] studied four systems that all purport to implement BM25 ranking and observed large differences in effectiveness. In fact, we have even observed instances where the same research group, using the same system, reported different baseline effectiveness values for the same ranking model, in different papers presented at the same conference.[1]

Without reproducible baselines, how can we trust experimental results, the validity of comparisons, and the conclusions drawn? Is the field actually making progress as a whole? The meta-analysis of Armstrong et al. [7] argued that there was little improvement in the effectiveness of ranking models for *ad hoc* retrieval from 1998 to 2008, and they identified comparisons to weak baselines as one of the culprits.

This work tackles the perennial problem of reproducible baselines in information retrieval research, focusing in particular on simple bag-of-words ranking models.[2] Although it would be desirable to have reproducible baselines that range in sophistication, for example, a phrase-based ranking model, a relevance feedback model, and a learning-to-rank baseline, in our opinion even reproducibility for these simple models has not been fully solved. Thus, this is where we start.

Our approach tries to simplify adoption and maximize impact by taking advantage of the open-source Java search engine Lucene. We present Anserini, a software toolkit built around Lucene that allows a researcher to reproduce results on a number of standard TREC test collections using a bevy of modern bag-of-words ranking models. This can be accomplished right out of the box, in three easy steps:

(1) Clone our Git source code repository at http://anserini.io/ and build the software package via a command (in Maven, the Java build manager) provided in our documentation.
(2) Issue one command to index a document collection of choice. The exact invocation can be copied-and-pasted directly from custom guides that we have created for each test collection.
(3) Issue another command to perform a retrieval run using topics from an existing test collection. The topics are included in our repository, and the exact invocations are also provided in the same guides above.

[1]We purposely omit citations here, because we do not feel it appropriate to single out any particular set of authors for criticism; our goal is to merely highlight issues with reproducibility that pervade the field.
[2]We are aware that what we call reproducibility in this article is perhaps better termed replicability or repeatability (see, for example, arguments by Drummond [21]). However, we have decided to stick with *reproducibility* to better reflect the current parlance of the information retrieval community.

The output is a retrieval run in standard TREC format that can be fed into a number of common evaluation tools such as `trec_eval` to reproduce the results reported in this article (the repository also includes qrels for scoring). Our study empirically validates three claims about Anserini (and by extension, Lucene):

(1) It is highly scalable and able to efficiently index large web collections.
(2) It provides modern bag-of-words ranking models that are just as effectiveness as implementations in Indri and Terrier, two popular open-source systems used by researchers today.
(3) It is fast in terms of providing low-latency query evaluation to support rapid *ad hoc* experimentation.

This article describes the efforts behind Anserini, focusing in particular on the software infrastructure that we have built around Lucene. Our system was originally described in a SIGIR 2017 short paper [57], but here we extend those experiments and provide additional details about our approach.

2 BACKGROUND AND RELATED WORK

Our work lies at the intersection of open-source information retrieval systems and reproducibility of information retrieval experiments. These two threads are related in that making source code available is touted across a broad range of scientific disciplines as one important step toward reproducibility [25, 44, 46]. In this section, we review previous work, including our own experiences.

2.1 Open-Source Information Retrieval Systems

Information retrieval researchers have a long history of developing and sharing systems to support their work, which can be traced back to Cornell's SMART system [13] from the mid 1980s.[3] Over the past several decades, various IR systems have been built to aid in the development of new retrieval models, to test hypotheses about information seeking, and to validate new evaluation methodologies. An incomplete list includes Lemur/Indri [38, 39], Galago [15], Terrier [35, 42], ATIRE [51], Ivory [30], JASS [31], MG4J [12], [60, 61]. Although some academic systems are widely used across many institutions (for example, Indri and Terrier), many researchers exclusively conduct experiments on their own systems, which contributes to difficulty in interpreting experimental results, since it is unclear if ranking models or implementations are actually being compared.

In 2005, a workshop on Open Source Web Information Retrieval (OSWIR) was held at the 2005 IEEE/WIC/ACM International Conferences on Web Intelligence & Intelligent Agent Technology [11]. A follow-up workshop on Open Source Information Retrieval (OSIR) was organized at SIGIR 2006, which provided a "forum that allow[ed] open source developers, consumers, and researchers to interact to coordinate their efforts" [58]. Another iteration of the workshop was held at SIGIR 2012 [50], which had similar goals. Both workshops were primarily venues for the exchange of information and featured lively discussion on future directions. Both the SIGIR 2006 and 2012 workshops included representatives from the Lucene community, who shared their experiences in cultivating an active community of users and contributors.

Academics typically build information retrieval systems to pursue some set of research objectives: for example, more effective ranking models, more efficient query evaluation, or better support for information seeking. The primary goal of these systems is to aid in the dissemination of research results. As such, academic codebases are usually not as well engineered as systems

[3]Personal communication, Chris Buckley.

designed for production deployment and frequently lack important features that are not directly needed to demonstrate an innovation. Common issues include poor documentation, lack of flexibility in supporting use cases not explored in a paper, inability to handle heterogeneous content, and limited support for parallelism and scaling. Furthermore, the growing complexity of modern software ecosystems and the resource constraints most academic research groups operate under make maintaining open-source systems a constant struggle, since much of these demands are not directly related to research.

In fact, there was a discussion about the increasing costs of running information retrieval experiments at the SIGIR 2012 workshop [50]:

> Discussion on the difficulty of performing search engine experiments under increasing collection sizes and decreasing budgets was had. For example, it is not trivial to obtain ClueWeb09, to index it, and to create a run for TREC. The hardware necessary is expensive and the skills in managing large collections are not easily obtained (debugging a program that takes many hours to run is time-consuming).

As a specific example, consider the ClueWeb12 collection,[4] which contains 733 million web pages totaling 5.54TB compressed (or 27.3TB uncompressed). The standard practice for working with this collection, as exemplified by the infrastructure built for the TREC 2014 Session Track [14], is to separately index partitions of the collection and then build a distributed broker architecture that integrates results from each partition. In this case, the organizers built 20 segment indexes that were distributed across eight servers.[5] The end-to-end system, they wrote, could be "quite slow, requiring 30 seconds or more to respond to some queries." In general, working with web-scale collections using existing academic systems can be time- and resource-intensive (not every research group can afford a cluster), even for basic tasks. It should not be this hard.

Today, only a small number of companies—mostly commercial web search engines such as Google and Bing—deploy their own custom infrastructure for search. For everyone else in industry, the open-source Lucene search engine (including other software in its ecosystem such as Solr and Elasticsearch) has become the *de facto* platform for building and deploying search applications. Prominent users include Apple, Bloomberg, Disney, LinkedIn, Netflix, and Twitter, as well as many online retailers and companies in financial services. In addition to broad adoption, Lucene benefits from a vibrant open-source community of developers. However, for various reasons that we discuss in Section 3, Lucene is not well-suited to supporting information retrieval research. We aim to address this issue.

More recently, Azzopardi et al. organized Lucene4IR,[6] a workshop "that brought together researchers and developers to discuss, plan, and develop a common set of teaching and training resources for students and researchers wishing to use Lucene for information retrieval research" [10]. The event included tutorials on common operations with Lucene (i.e., indexing, retrieval, etc.) as well as overviews of more advanced features such as extracting raw term statistics, implementing different scoring functions, and so on.

The Lucene for Information Access and Retrieval Research (LIARR) Workshop at SIGIR 2017 [9] represented the latest effort by information retrieval researchers to promote the use of Lucene for research. The workshop was not organized as a traditional "mini conference" but was rather designed as a hackathon for attendees to work hands-on with Lucene in a collaborative environment. For the participants, most of the workshop was spent in self-organized groups that explored

[4] http://www.lemurproject.org/clueweb12/.
[5] Personal communication, Ben Carterette.
[6] https://sites.google.com/site/lucene4ir/home.

various aspects of Lucene internals, ranging from the implementation of new retrieval features to gaining a better understanding of Lucene's document-scoring inner loop.

As co-organizers of the LIARR workshop, we are motivated by the desire to better align information retrieval research with the practice of building real-world search applications. We believe that a better alignment between academia and industry can lead to greater reproducibility of research results, richer collaborations, and more efficient knowledge transfer. The focus on Lucene maximizes our impact.

2.2 Reproducibility in Information Retrieval

Reproducibility is one of the fundamental pillars of the scientific method, plausibly dating back to Aristotle, and its associated challenges are certainly not limited to information retrieval. Just focusing on the computational sciences (in particular, setting aside the medical sciences), there has already been much discussion [21, 24, 37, 44] as well as concrete proposals to improve the state of affairs [20, 25, 46].

Within the information retrieval community, the 2009 landmark "improvements that don't add up" paper by Armstrong et al. [7] provided a meta-analysis that anchors recent discussions about reproducibility. The authors performed a longitudinal survey of over 100 papers from SIGIR and CIKM from 1998 to 2008 to track improvements in retrieval effectiveness on a wide range of commonly-used TREC test collections. Somewhat depressingly, they concluded:

> Most worryingly of all, there is no discernible upward trend in Ad-Hoc scores over time. Rather, the pattern is of researchers consistently reporting similar improvements over similar baseline scores, with results reported in 2008 generally indistinguishable from those reported in 1999. Matters are slightly better for the (more recent) Web collections, but even so there is no consistent upwards trend.

That is, progress in improved ranking models (at least according to the test collections examined) was mostly illusory during the decade from 1998 to 2008. Armstrong et al. diagnosed the problem primarily as researchers reporting "significant" improvements over weak baselines.

The authors also tackled a common defense: that most research groups do not actually aim to produce the best possible ranking in an absolute sense. Instead, they introduce a particular innovation, whose effectiveness is demonstrated using a paired experiment "with" and "without" that particular innovation. In these cases, the contribution is an examination of the innovation in isolation, and thus little attention is typically paid to the absolute effectiveness. As a response to this line of argumentation, Armstrong et al. considered the question of whether improvements over weak baselines are meaningful, even if they are statistically significant. That is: "How confident are we that a technique that yields an improvement over a weak baseline would also give an improvement over a strong one, and therefore be a worthwhile addition to state of the art systems?" They experimentally showed that weak improvements are *not* additive, a finding that has been subsequently confirmed by Kharazmi et al. [26]—hence, improvements that do not add up.

The proposed solution of Armstrong et al. in addressing the experimental failings of researchers is a repository of experimental results they called EvaluatIR [6]. In their proposal, researchers would upload runs using standard test collections along with appropriate metadata to a central repository. This resource would allow other researchers to quickly find the best-known results on a particular test collection and would enable referees to more effectively evaluate claims made in research papers. Better data management for information retrieval experiments is of course not a new idea [3], and a group of researchers organized the Data infrastructurEs for Supporting Information Retrieval Evaluation (DESIRE) Workshop at CIKM 2011 as a forum to coordinate and promote ongoing efforts [2].

Notably absent from EvaluatIR was a mechanism for depositing, managing, or executing code associated with the experimental runs, but this limitation has been addressed by subsequent work: VIRLab [22] provides a web-based virtual lab environment, primarily designed for students learning about information retrieval. The RISE platform [56] provides a Docker-managed environment for experimenting with different retrieval models on top of Indri. However, it is unclear if any of the above efforts have received significant uptake within the community and have fundamentally improved experimental practices in information retrieval.

Armstrong et al. [7] concluded their paper with a concrete challenge: "Let us build a public system that matches the BM25 run in the 1994 TREC-3 experiment, and then add to it the fruits of the past 15 years' research, to form a new baseline against which future effectiveness improvements can be properly measured." An initial effort along exactly these lines was spearheaded by one of the co-authors of this article as part of the workshop on Reproducibility, Inexplicability, and Generalizablity of Results (RIGOR) [5] at SIGIR 2015. The initial exercise for the workshop was subsequently expanded into the "open-source reproducibility challenge" [29], in which developers of various open-source search engines were invited to contribute reproducible runs from their systems in a common cloud-based environment. In total, seven systems participated, and the effort yielded a repository[7] that contains all code and scripts necessary to reproduce a wide range of *ad hoc* retrieval runs on the Gov2 test collection. In addition to standard bag-of-words baselines such as BM25, contributions included approximate ranking methods that trade effectiveness for efficiency along with slower but more effective proximity-based ranking models and techniques that incorporate relevance feedback. Beyond the repository itself, the exercise was valuable in identifying many unanticipated challenges, including the surprising amount of effort necessary to distill reproducible baselines into end-to-end execution scripts. For example, the effort revealed hidden dependencies in one system—a previously unpublished data processing script that made reproduction nearly impossible—and exposed two bugs in another system that were subsequently fixed.

This work is a direct extension of the reproducibility challenge discussed above by one of the co-organizers. Lucene was one of the systems evaluated, and it ranked fourth in terms of query latency. However, it was more effective than all the systems that were faster than it. Overall, Lucene represented a good tradeoff between effectiveness and efficiency and also demonstrated good indexing scalability. Given all the advantages discussed in the previous section, it made sense to focus additional development efforts around Lucene.

3 ANSERINI OVERVIEW

Despite its popularity in industry and broad adoption for operational search deployments in production environments, Lucene remains under-utilized in information retrieval research. This section begins with some high-level discussions of why we believe this might be the case to motivate our efforts in building Anserini. We then describe the additional software infrastructure we have built on top of Lucene to simplify *ad hoc* experimentation with standard information retrieval test collections.

3.1 Why Not Lucene?

From its very beginnings in 1999, Lucene was written for "real-world" search applications, not with researchers in mind. For the most part, its developers targeted an audience that mostly used search engines as black boxes, as opposed to researchers who require access to ranking internals such as scoring functions, mechanisms for postings traversal, and so on. Because of the target user

[7] https://github.com/lintool/IR-Reproducibility.

population, documentation for Lucene internals has always been quite poor, especially in keeping up with the rapid pace at which the developer community has been releasing improved versions of the system. Access to these internals is exactly what information retrieval researchers need for their studies, and therefore poor documentation has been a barrier to entry.

To further compound this issue, the internal APIs in Lucene are not organized in an intuitive manner, with awkwardly-named classes and many levels of indirection that make understanding the codebase difficult. As an example, to access the terms in a particular index (i.e., the dictionary), the developer needs to first open up an `IndexerReader`, access a `LeafReader` through a `LeafReaderContext`, select the desired field to obtain the `Terms` object, which provides an iterator over `TermsEnum`, which supplies a sequence of `BytesRef` objects, each of which finally provides access to the raw `byte[]` that represents the term (UTF-8 string). The `TermsEnum` is actually a `BytesRefIterator`, but the relationship between "terms" and "bytesref" is not immediately obvious. Given a term in the dictionary, the developer can now go back to the `LeafReader` and request its postings, which is represented by the `PostingsEnum` object, which implements the `DocIdSetIterator` interface. However, this interface doesn't conform to a standard iterator. For example, the termination condition (i.e., to identify the end of the iteration) is to check that the result of the `nextDoc()` method is equal to `DocIdSetIterator.NO_MORE_DOCS`. Thus, the `nextDoc()` method has overloaded semantics, simultaneously supplying the next document id as well as signaling when the caller has reached the end of the postings list. Finally, the developer has to remember to call `DocIdSetIterator.nextDoc()` on initialization before any of the per-document methods are available.

None of these issues are insurmountable for a competent developer, but together they contribute to a steep learning curve. To a core Lucene contributor steeped in the history of how the project evolved, there is likely good rationale for why the classes are designed and implemented (or evolved) this way. However, for researchers encountering Lucene for the first time, they are left with the impression that low-level APIs in Lucene are difficult to use. Note that for "black box" users of Lucene, who likely do not need access to system internals, these issues do not present hurdles to adoption.

Another side effect of Lucene's focus on "black box" search is that it has severely lagged behind in the implementation of modern ranking functions. For the longest time, the default scoring model was an *ad hoc* variant of TF-IDF. For example, Turtle et al. [54] showed that Lucene's out-of-the-box effectiveness was far worse than Indri's. Okapi BM25 was not added to Lucene until 2011,[8] more than a decade after it gained widespread adoption in the research community as being more effective than TF-IDF variants. This lag in adopting "research best practices" has contributed to the perception that Lucene is not effective and ill-suited for information retrieval research. Whatever the case might have been historically, this perception is no longer accurate today. Lucene comes with implementations of many modern bag-of-words ranking models, and we show that the effectiveness of Lucene's implementations is on par with those of Indri and Terrier, two popular systems used in academic research (see Section 4.3).

Finally, because Lucene is implemented in Java, there is sometimes the perception that it is slow and inefficient. Developers often point to the managed memory environment of the Java Virtual Machine (JVM) as not being conducive to efficient low-level implementations of search engine internals. However, in the reproducibility study of Lin et al. [29], Lucene was the fourth fastest system (out of seven) in terms of query latency, and it was more effective than all the systems that were faster than it. The study showed that while some configurations of Indri and Terrier were more effective, they were also much slower. Our experiments consider a much broader range of test

[8] https://issues.apache.org/jira/browse/LUCENE-2959.

collections and confirm these basic findings. The open-source community has devoted substantial effort to optimizing the performance of Lucene, and today the system is effective in producing high-quality rankings, efficient in supporting low-latency query evaluation, and scalable in rapidly indexing large web collections.

3.2 Simplifying *Ad Hoc* Experimentation

Test collections play a central role in information retrieval research and a substantial amount of research activity focuses on improving *ad hoc* retrieval using these evaluation resources. It is the explicit goal of Anserini to simplify *ad hoc* experimentation using Lucene to facilitate this core research activity.

The biggest hurdle to using Lucene for information retrieval research is perhaps best characterized by Grant Ingersoll, a Lucene committer as well as the CTO and co-founder of Lucidworks, a company that provides commercial Lucene products and support. He stated that "Lucene is a kit of parts" in the keynote presentation at the SIGIR 2017 LIAAR Workshop [9], and it does not prescribe how one would assemble those parts for various applications. For production search deployments, Solr fills this void, but no such "canonical assembly" exists specifically for information retrieval research. Anserini fills this gap.

To give a few specific examples: Lucene provides a rich set of APIs and implementations for document processing (parsing document file formats, tokenization, stopword removal, etc.) prior to indexing. Although the information retrieval community has developed various bits of code for ingesting standard test collections (e.g., the TREC SGML format, TREC-specific WARC formats for web collections, etc.), there did not exist a single repository that organizes all existing implementations in a unified manner, connected directly to appropriate Lucene APIs. Anserini provides exactly this—a unified document processing pipeline built on top of Lucene that is able to ingest standard test collections.

Another important contribution of Anserini is in scaling up multi-threaded indexing, which is important for handling large web collections. Although Lucene's indexing primitives are thread safe, it does not provide specific guidance on how to write a robust, high-throughput, multi-threaded indexer, where we need to deal with issues such as coordination, synchronization, and load balancing across multiple threads. In addition, Lucene provides a myriad of index configuration options, whose impact on performance is not transparent. Ultimately, what researchers desire is an end-to-end indexing application that ingests a document collection and builds an inverted index. Anserini provides exactly this. After some exploration, we settled on a multi-threaded implementation based on a work pool that has a user-configurable number of threads. Test collections are typically divided into files (which Anserini calls "file segments"): the indexer queues the processing of each file segment in the work pool, where they are indexed by individual threads. Anserini also provides default configurations (e.g., setting of buffer sizes) based on light tuning and our own experiences in working with a variety of test collections.

On the retrieval end, since Lucene was not originally designed for researchers, support for running experiments on standard test collections was largely missing. Anserini fills this gap by implementing a number of missing features: parsers for different query formats and a unified driver program for *ad hoc* experiments that outputs standard TREC format, which can then be evaluated using `trec_eval` or other common tools.

Putting everything together, Anserini allows researchers to conduct *ad hoc* experiments on a broad range of test collections right out of the box. After cloning our source code repository and building the package, one command builds an inverted index for a document collection, and a second command performs a retrieval run, ready for evaluation using standard tools. All topics and qrels, as well as the exact invocations to reproduce baseline runs, are provided in our detailed

guides for a diverse set of TREC test collections. If Lucene can be characterized as a "kit of parts," then Anserini provides a "canonical assembly" for information retrieval researchers. Beyond the codebase itself, we provide documentation to ease the learning curve for information retrieval researchers who desire access to Lucene internals.

3.3 Beyond Bag-of-Words Ranking Models: Multi-Stage Ranking Architectures

Lucene currently supports a number of modern bag-of-words ranking models (which we evaluate in Section 4.3) and Anserini simplifies *ad hoc* experimentation using them. The combination provides reproducible baselines for information retrieval researchers, which is the focus of this work.

What about reproducibility beyond simple bag-of-words ranking models? Although we have not explicitly addressed this (next) challenge, we believe that baseline bag-of-words ranking models also have an important role to play in modern multi-stage ranking architectures, which have been explored by researchers [8, 16, 17, 36, 49, 55] and deployed commercially [33, 43]. The basic idea is to decompose document ranking into a sequence of stages. The first stage consults the inverted index to identify a set of candidate documents based on a lightweight scoring model such as BM25 (or even Boolean keyword matching). One or more subsequent stages then processes these candidates, prunes away those not likely to be relevant, and reranks the remaining candidates. The intuition behind this multi-stage design is that later stage rankers can apply increasingly expensive features and richer ranking models without compromising query latency, since they are applied to fewer and fewer candidate documents. Learning-to-rank techniques [28], and more recently, neural ranking models, are typically deployed as later stage rankers in such multi-stage ranking architectures. For example, Bhaskar and Craswell [40] call this "telescoping" in their overview of neural models for information retrieval.

A consequence of this design is that research at the forefront of document ranking today still often relies on a candidate generation stage that uses a bag-of-words ranking model like BM25. We believe that Anserini can fill this role, and to this end we have designed a simple API that can be used to build reranking stages in a multi-stage ranking architecture. Anserini provides a reference implementation of the RM3 variant [1] of relevance models [27] using our reranking API. Thus, beyond providing reproducible baselines for *ad hoc* experiments, Anserini offers a platform for building end-to-end systems that implement state-of-the-art document ranking models. See Tu et al. [53] and Sequiera et al. [48] for two recent attempts by our team at integrating Lucene with neural ranking models.

4 EVALUATION

Anserini simplifies *ad hoc* experimentation and allows researchers to easily reproduce results with modern bag-of-words ranking models on diverse test collections. We describe experiments to support three specific claims about Anserini (and Lucene by extension) in this context. First, it is highly scalable and able to efficiently index large web collections. Second, the effectiveness of diverse ranking models is on par with results reported in previous research papers as well as implementations in Indri and Terrier. Finally, Anserini provides low-latency query evaluation to support rapid experimentation.

4.1 Experimental Setup

Our experiments used the following document collections:

- TREC Disks 1 & 2 (Disk12) and TREC Disks 4 & 5 (Disk45), excluding Congressional Record: newswire collections used in many TREC *ad hoc* evaluations.

Table 1. Topics and Document Collections Used in Our Experiments

Description	Topics	Document collection	# docs
Ad hoc task at TREC-1	51–100 (ad hoc)	Disks 1–2	741,616
Ad hoc task at TREC-2	101–150 (ad hoc)		
Ad hoc task at TREC-3	151–200 (ad hoc)		
Ad hoc task at TREC-6	301–350	Disks 4–5 (–Congressional Record)	528,030
Ad hoc task at TREC-7	351–400		
Ad hoc task at TREC-8	401–450		
Robust track at TREC 2004	601–700		
Robust track at TREC 2005	Robust04 hard queries	AQUAINT	1,031,455
Web task at TREC-9 (2001)	451–550	WT10g	1,688,402
Terabyte track at TREC 2004	701–750	Gov2	25,172,934
Terabyte track at TREC 2005	751–800		
Terabyte track at TREC 2006	801–850		
Web track at TREC 2010	51–100 (web)	ClueWeb09b	50,220,189
Web track at TREC 2011	101–150 (web)	ClueWeb09	503,892,800
Web track at TREC 2012	151–200 (web)		
Web track at TREC 2013	201–250 (web)	ClueWeb12-B13	52,249,039
Web track at TREC 2014	251–300 (web)	ClueWeb12	731,705,088

- The AQUAINT Corpus of English News Text, a newswire collection used in later TREC *ad hoc* evaluations.
- The WT10g and Gov2 web research collections from CSIRO (Commonwealth Scientific and Industrial Research Organisation), distributed by the University of Glasgow.
- ClueWeb collections from 2009 and 2012, which are web crawls gathered by Carnegie Mellon University.

For measuring effectiveness, we used topics and relevance judgments from a number of TREC *ad hoc* and web evaluations. The exact experimental conditions are shown in Table 1.

All experiments were conducted on an otherwise idle server with dual Intel Xeon E5-2699 v4 processors (2.2GHz, 22 cores, 55M cache) and 1TB RAM (16 × 64GB quad rank LRDIMMs). The server contains 18 × 10TB 7.2k SATA disks in a RAID-6 configuration running XFS. Overall, we can characterize this machine as "high-end commodity" but not beyond the reach of many information retrieval research groups. In terms of software configuration, our server runs RHEL 6.9 and JVM 1.8.0_141. Anserini is available at http://anserini.io/ and the current release is v0.1.0, based on Lucene 6.3.0.

4.2 Indexing Performance

The indexing performance of Anserini on the various collections is shown in Table 2. In all cases, we used 44 threads to saturate the cores of our test machine, although for the smaller collections indexing time with fewer threads is about the same or even slightly less (due to parallelism overhead). Stemming was performed using the Porter stemmer with a custom stopwords list.[9]

[9]See Lucene's `StandardAnalyzer` for details.

Table 2. Indexing Performance Comparing Different Configurations of Anserini

	Anserini (count)		Anserini (pos)		Anserini (all)	
Collection	time	size	time	size	time	size
Disk12	20s	217MB	38s	1.3GB	45s	2.5GB
Disk45	17s	179MB	33s	1.1GB	40s	2.1GB
AQUAINT	30s	329MB	52s	2.0GB	1m 7s	3.8GB
WT10g	1m 33s	741MB	2m 0s	2.0GB	2m 20s	9.3GB
Gov2	16m 2s	12GB	33m 28s	83GB	38m 27s	221GB
ClueWeb09b	43m	28GB	1h 19m	178GB	1h 41m	692GB
ClueWeb09	7h 45m	257GB	12h 10m	1.6TB	13h 44m	5.9TB
ClueWeb12-B13	51m	29GB	1h 31m	176GB	2h 28m	872GB
ClueWeb12	15h 17m	380GB	21h 27m	2.4TB	29h 54m	12TB

We report results for three different index configurations:

- count indexes where only term frequency information is stored (count),
- positional indexes where term position information is stored (pos),
- positional indexes that also store the parsed document vectors and the original documents in a forward index (all).

The "count" index condition is sufficient for bag-of-words queries. The "all" index condition supports the broadest range of operations: storing the parsed document vectors enables efficient relevance feedback and storing the raw documents allows snippet generation and browsing of original source documents. The "pos" index condition occupies a middle ground; these indexes support phrase queries and proximity operators, but not efficient relevance feedback or snippet generation. The tradeoff between these different index configurations is longer indexing time and greater space requirements, but with Anserini the developer has fine-grained control over exactly what is stored in the index.

For each index condition, we report the indexing time averaged over two trials as well as the index size computed with the Unix du command. Note that reported time units are different with the ClueWeb collections. Anserini is able to completely index TREC newswire collections in less than a minute. It is able to build count indexes over the smaller ClueWeb subsets (ClueWeb09b and ClueWeb12-B13) in less than an hour; storing the parsed document vectors and the original raw documents naturally takes longer. These two collections are roughly the same size in terms of number of documents, but web pages in ClueWeb12-B13 are on average larger, and therefore the longer indexing time for the "all" condition on ClueWeb12-B13 is likely dominated by the I/O costs of writing more data to disk.

Most impressively, Anserini is able to build a count index over *all* of ClueWeb12 (733 million pages totaling 5.54TB compressed) in around 15h. Even including positional information, parsed document vectors, and original raw documents, the entire index completes in about 30h, yielding an index of 12TB. These results show that Lucene achieves high-throughput indexing and has no trouble scaling to large web collections. Furthermore, Lucene is able to achieve this feat in a multi-threaded setting on a single server, without having to resort to distributed frameworks such as MapReduce, as with previous academic systems [15, 30].

To contextualize Lucene's indexing performance, we also indexed the same collections using Terrier 4.2.0 and Indri 5.11 on the same machine, following the same experimental procedure. Indri by default builds an index equivalent to the "all" Anserini index. For Terrier, we followed

Table 3. Indexing Performance of Terrier and Indri

Collection	Terrier		Indri	
	time	size	time	size
Disk12	2m 54s	184MB	4m 7s	2.4GB
Disk45	2m 28s	144MB	3m 6s	1.8GB
AQUAINT	5m 30s	276MB	6m 41s	3.6GB
WT10g	13m 53s	789MB	20m 45s	9.1GB
Gov2	5h 46m	11GB	13h 20m	199GB
ClueWeb09b	12h 27m	29GB	29h 10m	576GB
ClueWeb12-B13	16h 58m	36GB	53h 10m	700GB

a "best practices" guide written by the authors for ClueWeb09b,[10] which advocates use of the single-pass indexer.[11] For consistency, we used the same setting across all collections. This builds the equivalent of the Anserini "count" index. In both cases, stemming was performed with the Porter stemmer, although Indri[12] and Terrier[13] use different stopwords lists. We used the following memory allocations for both systems: 10g for Disk12, Disk45; 20g for AQUAINT; 50g for WT10g; 500g for Gov2; 1000g for ClueWeb09b and ClueWeb12-B13. Due to the long running times on ClueWeb09b and ClueWeb12-B13, we did not attempt to index all of ClueWeb09 and ClueWeb12.

Results from these indexing experiments are shown in Table 3. For comparable index configurations, we see that Lucene is much faster, which supports our empirical claim that Lucene is highly scalable and able to efficiently index large web collections.

4.3 Ranking Effectiveness and Efficiency

The main goal of this article is to demonstrate the suitability of Anserini for information retrieval research. From the perspective of effectiveness, we describe two sets of experiments: First, we conducted a series of reproducibility experiments comparing the effectiveness of Lucene's ranking models with figures reported in research papers that have examined those models. Anserini's effectiveness is higher in some cases and lower in other cases, but overall, we are able to reproduce results from the comparison papers. Second, for the same ranking models, we compared the effectiveness of Lucene against reimplementations in Indri (using the RISE framework [56]) and Terrier, across a number of standard TREC test collections. Overall, experiments show that the effectiveness of ranking models in Anserini is on par with those from Indri and Terrier.

Specifically, we examined the following ranking models, all of which have been implemented in Lucene:

- **BM25** [47], derived from the probabilistic retrieval framework, is one of the most commonly used baselines in information retrieval.
- **LM** refers to query likelihood with respect to Dirichlet-smoothed language models [59], derived from the language modeling framework.
- **PL2** [4] is a representative ranking model from the divergence from randomness framework. It measures the randomness of terms using Poisson distributions with Laplacian smoothing.
- **F2EXP** is from the family of axiomatic models [23], which starts with an existing ranking function (e.g., BM25) and searches for instantiations that satisfy more retrieval constraints.

[10] http://ir.dcs.gla.ac.uk/wiki/Terrier/ClueWeb09-B.
[11] Using the command `bin/trec_terrier.sh -i -j`.
[12] http://www.lemurproject.org/stopwords/stoplist.dft.
[13] https://github.com/terrier-org/terrier-core/blob/4.2/src/resources/stopword-list.txt.

- **SPL** is derived from an information-theoretic model [18] that attempts to capture relevance in terms of how a word deviates from its average behavior. For SPL, the smoothed power-law distribution is used.

Before presenting results, we discuss one final detail. Due to multi-threaded indexing, there is unavoidable non-determinism in the order that documents are added to the index. This has an impact on breaking score ties during retrieval: Lucene by default uses the internal document id assigned at index time, which means that different index instances of the same collection may yield slightly different rankings. We have separately explored this issue [32] and quantified the differences in effectiveness arising from this non-determinism to be rather small (typically in the fourth decimal place for average precision). Anserini addresses this issue by breaking score ties using collection document ids, and thus guarantees that runs using different index instances yield exactly the same results. However, this comes at a cost in terms of query evaluation latency due to the necessity of consulting external identifiers during the inner loop of query evaluation. For ClueWeb collections, query evaluation is around 20% slower compared to an arbitrary tie-breaking approach [32]. All results presented below were conducted under this strict reproducibility condition.

In our first set of experiments, we attempted to reproduce as closely as possible the settings used in research papers that examined the models in detail; note that these are not necessarily the papers in which the models were first proposed. Effectiveness in terms of average precision (AP) at rank 1000 is shown in Table 4 for various combinations of document collections and topics. The first column provides citations to the comparison paper for each ranking model. The "Reference" column contains figures copied directly from the comparison papers, and the "Anserini" column reports our reproduced results. Official TREC topics have three components—the title, description, and narrative—and the comparison research papers often describe experiments under different conditions, so our table is similarly organized to facilitate comparisons.

We see that overall, our reproduced results are quite comparable to those reported in the comparison research papers, with the exception of PL2 on title, description, and narrative. In some cases, the original results are higher and in other cases, Anserini results are higher. Note that since we are merely reporting effectiveness figures from prior published work, we lack the details for significance testing.

Nevertheless, these results support our claim that Lucene is suitable for information retrieval research, since it is able to support diverse ranking models. For the large PL2 differences, it is worth pointing out that Yang and Fang [56] observed similar discrepancies in their reproducibility setup with Indri. We suspect that for this model, small changes in preprocessing can lead to large differences in effectiveness.

In our second set of experiments, we compared the retrieval effectiveness of Indri (again, using the RISE framework [56]), Terrier, and Anserini using the same ranking models. For Terrier, we only included BM25, LM, and PL2, since they are the only ranking models tunable out of the box. In this setup, unlike in the previous experiments that tried to replicate experimental conditions in the reference papers as closely as possible, all the Indri and Terrier comparisons shared preprocessing and indexing configurations.

For all systems, we report results from parameter tuning to optimize average precision (AP) at rank 1000 on the newswire collections, WT10g, and Gov2, and NDCG@20 for the ClueWeb collections. There was no separation of training and test data, so these results should be interpreted as oracle settings. We explored the following parameter settings:

- for BM25, $k = 0.9$ and $b \in [0, 1]$ in increments of 0.1;
- for LM, $\mu \in [0, 5000]$ in increments of 500;
- for PL2, $c \in [0.5, 20]$ in increments of 0.5;

Table 4. Reproducibility Results under Different Collection/Topic Combinations, with Effectiveness Reported in Terms of Average Precision (AP) at Rank 1000

(a) "title" topics

Model	Collection	Topics (ad hoc)	Reference	Anserini	Δ
BM25 [34]	Disk45	301–450 601–700	0.2544	0.2529	−0.59%
	WT10g	451–550	0.1879	0.2013	+7.13%
	Gov2	701–850	0.2931	0.3049	+4.03%
LM [59]	Disk45	351–400	0.1860	0.1832	−1.51%
	Disk45	401–450	0.2560	0.2461	−3.87%
F2EXP [23]	Disk45	351–400	0.1870	0.1882	+0.64%
	Disk45	401–450	0.2570	0.2475	−3.70%
SPL [18]	Disk12	151–200	0.2620	0.2445	−6.68%
	Disk45	301–450 601–700	0.2540	0.2495	−1.77%

(b) "description" topics

Model	Collection	Topics (ad hoc)	Reference	Anserini	Δ
BM25 [34]	Disk45	301–450 601–700	0.2260	0.2256	−0.18%
	WT10g	451–550	0.1745	0.1870	+7.16%
	Gov2	701–850	0.2234	0.2380	+6.54%
LM [59]	Disk45	351–400	0.1820	0.1747	−4.01%
	Disk45	401–450	0.2280	0.2174	−4.65%
F2EXP [23]	Disk45	351–400	0.1860	0.1911	+2.74%
	Disk45	401–450	0.2360	0.2349	−0.47%

(c) "title + description + narrative" topics

Model	Collection	Topics (ad hoc)	Reference	Anserini	Δ
LM [59]	Disk45	351–400	0.2240	0.2125	−5.13%
	Disk45	401–450	0.2600	0.2447	−5.88%
PL2 [4]	Disk12	51–100	0.2065	0.2352	+13.90%
	Disk12	101–150	0.2383	0.2787	+16.95%
	Disk12	151–200	0.2705	0.2178	−19.48%
	Disk45	301–350	0.2569	0.1728	−32.74%
	Disk45	351–400	0.2212	0.1891	−14.51%
	Disk45	401–450	0.2562	0.2212	−13.66%
F2EXP [23]	Disk45	351–400	0.2250	0.2324	+3.29%
	Disk45	401–450	0.2600	0.2669	+2.65%

The "Reference" column contains figures copied from previous papers (sources denoted in the first column) and the "Anserini" column reports our reproduced results.

- for F2EXP $s \in [0, 1]$ in increments of 0.05;
- for SPL $c \in [0.5, 10]$ in increments of 0.5.

Effectiveness on various collection/topic combinations are shown in Table 5. We applied a paired two-tailed t-test ($p = 0.05$) to assess the significance of effectiveness differences between each pair of systems for each model; significant differences are denoted with superscript ^{A}, ^{I}, and ^{T}, indicating that the metric is significantly better than Anserini, Indri, and Terrier, respectively. Across these many conditions, we only observe two cases where effectiveness differences are

Table 5. Effectiveness Comparisons between Anserini, Indri, and Terrier on Standard TREC Test Collections

(a) Newswire, WT10g, and Gov2: Effectiveness measured in terms of average precision (AP) at rank 1000.

Collection		Disk12	Disk45	Disk45	AQUAINT	WT10g	Gov2
Topics (ad hoc)		51–200	301–450	601–700	Robust05	451–550	701–850
Classic	Anserini	0.1920	0.2037	0.2450	0.1494	0.1234	0.1564
BM25	Indri	0.2040	0.2226	0.2895	0.2041	0.1955	0.2970
BM25	Terrier	0.2286	0.2225	0.2882	0.2022	0.2136	0.3050
BM25	Anserini	0.2302	0.2298	0.2916	0.2090	0.2012	0.3030
LM	Indri	0.2269	0.2246	0.2900	0.1980	0.1915	0.2995
LM	Terrier	0.2231	0.2231	0.2902	0.2028	0.2111	0.2976
LM	Anserini	0.2250	0.2263	0.2893	0.2026	0.2034	0.2954
PL2	Indri	0.2279	0.2247	0.2932	0.1975	0.2012^{A}	0.3029
PL2	Terrier	0.2231	0.2257	0.2942	0.1969	0.2129^{AI}	0.3076
PL2	Anserini	0.2230	0.2263	0.2917	0.2006	0.1889	0.3067
F2EXP	Indri	0.2277	0.2277	0.2877	0.1951	0.2020	0.2840
F2EXP	Anserini	0.2221	0.2222	0.2795	0.1926	0.1932	0.2924
SPL	Indri	0.2240	0.2230	0.2917	0.1960	0.1947^{A}	0.3017
SPL	Anserini	0.2151	0.2250	0.2900	0.1969	0.1726	0.3070

(b) ClueWeb collections: Effectiveness measured in terms of NDCG@20.

Collection		ClueWeb09b	ClueWeb09	ClueWeb12-B13	ClueWeb12
Topics (web)		51–200	51–200	201–300	201–300
Classic	Anserini	0.0630	0.0201	0.0425	0.0619
BM25	Indri	0.1390	-	0.1306	-
BM25	Terrier	0.1456	-	0.1216	-
BM25	Anserini	0.1422	0.0974	0.1292	0.2380
LM	Indri	0.1164	-	0.1164	-
LM	Terrier	0.1287	-	0.1154	-
LM	Anserini	0.1227	0.0807	0.1178	0.2116
PL2	Indri	0.1223	-	0.1205	-
PL2	Terrier	0.1312	-	0.1188	-
PL2	Anserini	0.1274	0.0812	0.1206	0.2022
F2EXP	Indri	0.1321	-	0.1140	-
F2EXP	Anserini	0.1399	0.0914	0.1280	0.2330
SPL	Indri	0.1209	-	0.1196	-
SPL	Anserini	0.1273	0.0835	0.1206	0.2153

The superscript ^{A}, ^{I}, and ^{T} indicate significantly better than Anserini, Indri, and Terrier, respectively, based on a paired two-tailed t-test ($p = 0.05$).

significant (PL2 and SPL), both on WT10g. For both models, Anserini outperforms Indri on most of the other collections (although the differences are not significant).

We also report results from Lucene's default TF-IDF ranking function (denoted "Classic"), which in most cases is far less effective than any of the other ranking models we examined. This lends credence to the impression that Lucene does not provide effective rankings, at least prior to the implementation of BM25, LM, and more modern ranking models. However, it is clear today

Table 6. Ranking Efficiency (Average Query Latency) Comparing Anserini, Indri, and Terrier Using BM25 (Top 1000 hits)

(a) Newswire and Web collections

Collection	Disk12	Disk45	Disk45	AQUAINT	WT10g	Gov2
Topics (ad hoc)	51–200	301–450	601–700	Robust04	451–550	701–850
Anserini (count)	13ms	13ms	13ms	20ms	30ms	123ms
Anserini (pos)	13ms	13ms	13ms	20ms	30ms	127ms
Anserini (all)	60ms	47ms	48ms	60ms	100ms	267ms
Indri	32ms	23ms	26ms	22ms	34ms	512ms
Terrier	17ms	11ms	12ms	30ms	31ms	331ms

(b) ClueWeb collections

Collection	ClueWeb09b	ClueWeb09	ClueWeb12-B13	ClueWeb12
Topics (web)	51–200	51–200	201–300	201–300
Anserini (count)	0.29s	2.5s	0.28s	3.8s
Anserini (pos)	0.29s	2.3s	0.28s	3.9s
Anserini (all)	0.46s	2.8s	0.53s	4.5s
Indri	1.3s	-	1.1s	-
Terrier	0.63s	-	0.76s	-

that Lucene provides ranking models that are on par with those of Indri and Terrier in terms of effectiveness.

Finally, in Table 6, we report the results of efficiency experiments comparing the systems under the same collection/topic combinations as Table 5. For these experiments, we used BM25 to retrieve up to 1000 hits. The table reports *per query* retrieval latency (note different units of time). In all cases, we warmed up underlying operating system caches by first performing a few experimental runs and discarding the results. We then computed averages over the next three trials in each condition: these are the values reported in the table. For Anserini, latencies over all three index conditions are reported.

We make a few interesting observations: query latency with Anserini appears to differ based on the Lucene index configuration. For count and positional indexes, there is no measurable difference for small collections and sometimes a small difference for large collections. However, retrieval from the "all" condition (positional indexes plus document vectors and original documents) is noticeably slower. For small collections, Anserini is faster than Indri with count and positional indexes, but slower in the "all" index configuration. For anything larger than Gov2, Anserini is faster than Indri in all index configurations. Terrier and Anserini (count and positional indexes) have comparable query latencies on small collections, but all index configurations of Anserini are faster on large collections. These results dispel the myth that the Java Virtual Machine is not conducive to implementing efficient search engines. Indri (implemented in C++) is slower than Terrier (implemented in Java) for nearly all collections. Although the choice of language does matter, performance is more an issue of careful software engineering, an area where the Lucene community has made significant investments over the years.

In summary, our experiments show that Anserini is at least as good as Indri and Terrier in terms of effectiveness, and faster in both indexing and retrieval. These results are consistent with findings from the Open-Source IR Reproducibility Challenge [29]. Taken together, empirical evidence presents a compelling case for adopting Lucene for information retrieval research.

5 CONCLUSIONS

One motivation for having reproducible baselines, as discussed in the Introduction, is the proliferation of model variants that confound comparisons. What exactly does BM25 mean? In truth, Lucene's BM25 is not a perfectly faithful implementation of Robertson's original equations [47]. However, we argue that it doesn't actually matter: at the end of the day, rankings are not generated by ranking models—they are generated by implementations of those ranking models, rolled up with hundreds of small design decisions whose only "documentation" is the source code itself. Thus, what we *call* a baseline is not important as long as the label points to a stable computational artifact and we have a repeatable way to execute that computational artifact to generate results. We believe we've accomplished this with Anserini.

One potential downside of operationalizing reproducible baselines as computational artifacts is that the artifacts themselves evolve. The results reported in this article, for example, represent a snapshot in time—both for Anserini as well as Indri and Terrier. Anserini, in turn, depends on Lucene, which changes over time as features are added and bugs are fixed. We feel that such a coupling is acceptable for a few reasons: First, the Lucene community is large and diverse, and the contributors are generally cognizant of the downstream impact of changes that may affect the behavior of systems that build on Lucene. Second, Anserini's focus on reproducible baselines means that it is dependent mostly on core Lucene features, which are relatively mature and stable.

In our experience, concerns about shifting Lucene foundations have mostly proved unfounded: Since the inception of the project, we have gone through numerous Lucene upgrades, all of which have been successful with minimal effort. Furthermore, the simplicity and speed of running baseline retrieval experiments mean that we can run regression tests (part of our repository) on all existing test collections whenever the underlying Lucene version changes. We are confident that Anserini can continue to track the evolution of Lucene itself. Since the source code is hosted on GitHub, all code changes are easy to browse and access. If there is ever any doubt about which "version" of Anserini is used to perform a particular experiment, then one could unambiguously refer to a commit id, as advocated by Crane [19] in a recent reproducibility study on neural networks.

To conclude, our message to the information retrieval community is that Lucene is efficient and scalable without compromising effectiveness. Furthermore, Lucene has the benefit of a large user community and broad adoption in industry. Anserini provides the "canonical assembly" of Lucene's "kit of parts" to support *ad hoc* experimentation and reproducible baselines for information retrieval researchers. We hope that our system will help to better align the research and practice of information retrieval, accelerating progress in the field.

REFERENCES

[1] Nasreen Abdul-Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D. Smucker, Trevor Strohman, Howard Turtle, and Courtney Wade. 2004. UMass at TREC 2004: Novelty and HARD. In *Proceedings of the 13th Text REtrieval Conference (TREC'04)*.

[2] Maristella Agosti, Nicola Ferro, and Costantino Thanos. 2012. DESIRE 2011: Workshop on Data Infrastructures for Supporting Information Retrieval Evaluation. *SIGIR Forum* 46, 1 (2012), 51–55.

[3] Maristella Agosti, Giorgio Maria Di Nunzio, and Nicola Ferro. 2007. Scientific data of an evaluation campaign: Do we properly deal with them? In *Proceedings of the Conference on Evaluation of Cross-Language Information Retrieval Systems (CLEF'06)*. 11–20.

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

[5] Jaime Arguello, Matt Crane, Fernando Diaz, Jimmy Lin, and Andrew Trotman. 2015. Report on the SIGIR 2015 workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum* 49, 2 (2015), 107–116.

[6] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. EvaluatIR: An online tool for evaluating and comparing IR systems. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. 834.

[7] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *Proceedings of the 18th International Conference on Information and Knowledge Management (CIKM'09)*. 601–610.

[8] Nima Asadi and Jimmy Lin. 2013. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proceedings of the 36th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*. 997–1000.

[9] Leif Azzopardi, Matt Crane, Hui Fang, Grant Ingersoll, Jimmy Lin, Yashar Moshfeghi, Harrisen Scells, Peilin Yang, and Guido Zuccon. 2017. The Lucene for Information Access and Retrieval Research (LIARR) workshop at SIGIR 2017. In *Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17)*. 1429–1430.

[10] Leif Azzopardi, Yashar Moshfeghi, Martin Halvey, Rami S. Alkhawaldeh, Krisztian Balog, Emanuele Di Buccio, Diego Ceccarelli, Juan M. Fernández-Luna, Charlie Hull, Jake Mannix, and Sauparna Palchowdhury. 2017. Lucene4IR: Developing information retrieval evaluation resources using Lucene. *SIGIR Forum* 50, 2 (2017), 58–75.

[11] Michel Beigbeder and Wai Gen Yee. 2015. OSWIR 2005 Workshop, Final Report.

[12] Paolo Boldi and Sebastiano Vigna. 2005. MG4J at TREC 2005. In *Proceedings of the 14th Text REtrieval Conference (TREC'05)*.

[13] Chris Buckley. 1985. *Implementation of the SMART Information Retrieval System*. Department of Computer Science TR 85-686. Cornell University.

[14] Ben Carterette, Evangelos Kanoulas, Mark Hall, and Paul Clough. 2014. Overview of the TREC 2014 session track. In *Proceedings of the 23rd Text REtrieval Conference (TREC'14)*.

[15] Marc-Allen Cartright, Samuel Huston, and Henry Feild. 2012. Galago: A modular distributed processing and retrieval system. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*. 25–31.

[16] Ruey-Cheng Chen, Luke Gallagher, Roi Blanco, and J. Shane Culpepper. 2017. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17)*. 445–454.

[17] Charles L. A. Clarke, J. Shane Culpepper, and Alistair Moffat. 2016. Assessing efficiency–effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments. *Info. Retriev.* 19, 4 (2016), 351–377.

[18] Stéphane Clinchant and Eric Gaussier. 2010. Information-based models for ad hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*. 234–241.

[19] Matt Crane. 2018. Questionable answers in question answering research: Reproducibility and variability of published results. *Trans. Assoc. Comput. Linguist.* 6 (2018), 241–252.

[20] Jens Dittrich and Patrick Bender. 2015. Janiform intra-document analytics for reproducible research. *Proc. VLDB Endow.* 8, 12 (2015), 1972–1975.

[21] Chris Drummond. 2009. Replicability is not reproducibility: Nor is it good science. In *Proceedings of the 4th Workshop on Evaluation Methods for Machine Learning at ICML*.

[22] Hui Fang, Hao Wu, Peilin Yang, and ChengXiang Zhai. 2014. VIRLab: A web-based virtual lab for learning and studying information retrieval models. In *Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14)*. 1249–1250.

[23] Hui Fang and ChengXiang Zhai. 2005. An exploration of axiomatic approaches to information retrieval. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*. 480–487.

[24] Nicola Ferro, Norbert Fuhr, Kalervo Järvelin, Noriko Kando, Matthias Lippold, and Justin Zobel. 2016. Increasing reproducibility in IR: Findings from the Dagstuhl seminar on "reproducibility of data-oriented experiments in e-science." *SIGIR Forum* 50, 1 (2016), 68–82.

[25] Bill Howe. 2012. Virtual appliances, cloud computing, and reproducible research. *Comput. Sci. Eng.* 14, 4 (2012), 36–41.

[26] Sadegh Kharazmi, Falk Scholer, David Vallet, and Mark Sanderson. 2016. Examining additivity and weak baselines. *ACM Trans. Info. Syst.* 34, 4 (2016), Article No. 23.

[27] Victor Lavrenko and W. Bruce Croft. 2001. Relevance-based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)*. 120–127.

[28] Hang Li. 2014. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers.

[29] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. 2016. Toward reproducible baselines: Open-Source IR Reproducibility Challenge. In *Proceedings of the 38th European Conference on Information Retrieval (ECIR'16)*. 408–420.

[30] Jimmy Lin, Donald Metzler, Tamer Elsayed, and Lidan Wang. 2009. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. In *Proceedings of the 18th Text REtrieval Conference (TREC'09)*.

[31] Jimmy Lin and Andrew Trotman. 2015. Anytime ranking for impact-ordered indexes. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR'15)*. 301–304.

[32] Jimmy Lin and Peilin Yang. 2018. Repeatability corner cases in document ranking: The impact of score ties. *arXiv:1807.05798*.

[33] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'17)*. 1557–1565.

[34] Yuanhua Lv and ChengXiang Zhai. 2011. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*. 7–16.

[35] 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*. 60–63.

[36] Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*. 437–444.

[37] Jill P. Mesirov. 2010. Accessible reproducible research. *Science* 327, 5964 (2010), 415–416.

[38] Donald Metzler and W. Bruce Croft. 2004. Combining the language model and inference network approaches to retrieval. *Info. Process. Manage.* 40, 5 (2004), 735–750.

[39] Donald Metzler, Trevor Strohman, Howard Turtle, and W. Bruce Croft. 2004. Indri at TREC 2004: Terabyte track. In *Proceedings of the 13th Text REtrieval Conference (TREC'04)*.

[40] Bhaskar Mitra and Nick Craswell. 2017. Neural models for information retrieval. *arXiv:1705.01509v1*.

[41] Hannes Mühleisen, Thaer Samar, Jimmy Lin, and Arjen de Vries. 2014. Old dogs are great at new tricks: Column stores for IR prototyping. In *Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14)*. 863–866.

[42] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the SIGIR 2006 Workshop on Open Source Information Retrieval*.

[43] Jan Pedersen. 2010. Query understanding at Bing. In *Industry Track Keynote at the 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*.

[44] Roger D. Peng. 2011. Reproducible research in computational science. *Science* 334, 6060 (2011), 1226–1227.

[45] Jay M. Ponte and W. Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*. 275–281.

[46] Karthik Ram. 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code Biol. Med.* 8, 7 (2013).

[47] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of the 3rd Text REtrieval Conference (TREC'94)*. 109–126.

[48] Royal Sequiera, Gaurav Baruah, Zhucheng Tu, Salman Mohammed, Jinfeng Rao, Haotian Zhang, and Jimmy Lin. 2017. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. In *Proceedings of the SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR'17)*.

[49] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and effective retrieval using selective pruning. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM'13)*. 63–72.

[50] Andrew Trotman, Charles L. A. Clarke, Iadh Ounis, Shane Culpepper, Marc-Allen Cartright, and Shlomo Geva. 2012. Open Source Information Retrieval: A report on the SIGIR 2012 workshop. *SIGIR Forum* 46, 2 (2012), 95–101.

[51] Andrew Trotman, Xiang-Fei Jia, and Matt Crane. 2012. Towards an efficient and effective search engine. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*. 40–47.

[52] Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to BM25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium (ADCS'14)*. 58–66.

[53] Zhucheng Tu, Matt Crane, Royal Sequiera, Junchen Zhang, and Jimmy Lin. 2017. An exploration of approaches to integrating neural reranking models in multi-stage ranking architectures. In *Proceedings of the SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR'17)*.

[54] Howard Turtle, Yatish Hegde, and Steven A. Rowe. 2012. Yet another comparison of Lucene and Indri performance. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*. 64–67.

[55] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. 105–114.

[56] Peilin Yang and Hui Fang. 2016. A reproducibility study of information retrieval models. In *Proceedings of the 2nd ACM International Conference on the Theory of Information Retrieval (ICTIR'16)*. 77–86.

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

[58] Wai Gen Yee, Michel Beigbeder, and Wray Buntine. 2006. SIGIR06 workshop report: Open source information retrieval systems (OSIR06). *SIGIR Forum* 40, 2 (2006), 61–65.

[59] Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Info. Syst.* 22, 2 (2004), 179–214.

[60] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. 2010. *Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press.

[61] Bodo Billerbeck, Adam Cannane, Abhijit Chattaraj, Nicholas Lester, William Webber, Hugh E. Williams, John Yiannis, and Justin Zobel. 2004. RMIT University at TREC 2004. In *Proceedings of the 13th Text REtrieval Conference (TREC'04)*.

Received October 2017; revised April 2018; accepted July 2018