

Geese Belong in (Data) Lakes: An Index Lake for Anserini

Gijs Hendriksen
Radboud University
Nijmegen, The Netherlands
gijs.hendriksen@ru.nl

Djoerd Hiemstra
Radboud University
Nijmegen, The Netherlands
djoerd.hiemstra@ru.nl

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

Arjen P. de Vries
Radboud University
Nijmegen, The Netherlands
arjen.devries@ru.nl

Abstract

We present an *index lake* for Anserini: a remote index representation of all inverted indexes from Anserini’s reproducibility suite. Each index is represented as a set of Parquet files on remote object storage, and retrieval is performed through SQL with DuckDB. Through an extensive benchmark of 239 experiments across 50 collections, we empirically confirm our results are consistent with Anserini’s. We analyze query latencies over the index lake and conclude that (1) most experiments can be run relatively efficiently, requiring only a couple of seconds per query; and (2) for large collections and learned sparse indexes, queries suffer from increased latency due to high overhead. We are also able to reduce total network transfer, since (1) index lakes are smaller than the corresponding prebuilt Lucene indexes; and (2) up to 90% of the index can be skipped by pushing filters on the postings table to the data lake. The index lake increases reproducibility and makes it easier to run new experiments on standard test collections and prebuilt indexes.

CCS Concepts

• Information systems → Information retrieval.

Keywords

Index Lakes, Reproducibility, Anserini

ACM Reference Format:

Gijs Hendriksen, Djoerd Hiemstra, Jimmy Lin, and Arjen P. de Vries. 2026. Geese Belong in (Data) Lakes: An Index Lake for Anserini. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM ’26)*, November 07–11, 2026, Rome, Italy. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3799682.3840188>

1 Introduction

The inverted index is the data structure at the heart of modern retrieval systems. Lately, researchers have started sharing indexes built for their experiments with the community [7, 8]. This serves two purposes: (1) by sharing the index, other research groups can reproduce the experimental findings of a paper much more easily, and (2) the prebuilt index can be used for follow-up experiments by other groups. This reuse reduces the entry-level cost of getting started with a new collection, and promotes Green IR [12]. The latter is especially important with the ever-increasing number of test collections (with different purposes like question answering or retrieval-augmented generation) and especially the rise of costly neural retrieval models like dense or learned sparse retrieval (LSR).

The Anserini [16] team has built an extensive reproducibility suite around standard IR test collections and popular retrieval models, sharing prebuilt Lucene indexes for many collections from the MS MARCO [10], BEIR [15] and BRIGHT [14] benchmarks (as well as the smaller CACM test collection [1]). This includes lexical indexes used for models like BM25, but also vector indexes for dense retrieval and inverted indexes with quantized LSR representations. Still, running the reproducibility experiments (and running follow-up experiments with new query sets) requires downloading the full index and processing the queries with Anserini. For most topic sets, the overhead of downloading the index remains costly, given that usually most terms in the index do not occur in any of the queries.

In this paper, we introduce a different way of using Anserini’s prebuilt indexes. We build on the idea of the remote index [2], in which inverted indexes are stored as Parquet files on remote object storage. Each inverted index shared by Anserini is converted into this generic format and shared in a public S3 bucket, turning any analytical query engine that can read Parquet files from remote object storage into an advanced retrieval system. To make the indexes easy to use, we expose them publicly as a DuckLake (the data lake format developed by the DuckDB team [11]). The data lake includes the topics (both raw and pretokenized) and associated qrels for each collection, and exposes macros for simplified retrieval (see Section 2.4). We refer to such a collection of remote indexes, distributed through a data lake architecture, as an *index lake*.

In our case, we release a *public* index lake for Anserini, which in total contains 164 inverted indexes for 50 collections. In a series of 239 experiments, we empirically verified that our evaluation results are consistent with those of Anserini’s reproducibility benchmark. In the rest of the paper, we describe the index lake (including how it can be used) and show how it can reduce the barrier for running IR experiments – both in ease-of-use and in reduced network traffic.¹

2 The Index Lake

2.1 Inverted Indexes

To recreate Anserini’s reproducibility benchmark as a data lake, we downloaded all prebuilt Lucene indexes from Anserini’s Hugging Face datasets page.² This includes lexical indexes (denoted by Anserini as *inverted*) and inverted indexes with quantized impact scores for models like SPLADE-v3 [5] and uniCOIL [6] (denoted as *impact*). We disregard positional indexes, as Anserini does not use positions for sparse retrieval in its reproducibility benchmark. Finally, dense retrieval is hard to frame efficiently in terms of remote queries, as it requires access to all vectors of potentially relevant



This work is licensed under a Creative Commons Attribution 4.0 International License. *CIKM ’26, Rome, Italy*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2539-5/2026/11
<https://doi.org/10.1145/3799682.3840188>

¹Instructions on using the index lake can be found in our Git repository: <https://gitlab.science.ru.nl/informagus/anserini-index-lake>

²<https://huggingface.co/castorini/datasets>; download URLs were extracted from <https://github.com/castorini/anserini/tree/81181940/src/main/resources/prebuilt-indexes>.

Table 1: Tables included in the index lake. Each table consists of several Hive-partitioned Parquet files, where each partition contains a single Parquet file. Individual Parquet files are sorted by the columns in bold.

Table name	Partition columns	Parquet file columns	#files	#rows
documents	collection	docid , id	50	476.8 M
postings	collection, representation, field	term , docid , tf	189	177.2 B
lengths	collection, representation, field	docid , length	112	1.1 B
topics	collection, topics_key, representation	qid , query	263	408.2 K
qrels	collection, eval_key	qid , id, label	82	3.1 M
exclusions	collection, topics_key	qid , id	3	1.3 M

documents. Therefore, we do not include Anserini’s vector indexes, but we do intend to revisit dense retrieval in future work.

The downloaded indexes were transformed into the Parquet-based remote index format [2], creating a single Parquet file for each field in the index. To ensure we could reproduce Anserini’s evaluation results, Lucene’s document lengths were also exported (for the lexical indexes). For collections with multiple indexes (e.g. a lexical and an LSR index, or a flat and a multifield index), internal document identifiers were reassigned to be consistent across indexes, such that multiple retrieval models could be combined during ranking without having to look up the TREC ID intermediately.

2.2 Test Collections

Anserini also includes topics and qrels for a large number of standard test collections. To enable full reproducibility using the index lake only, it also includes these topics and relevance assessments.³ Unfortunately, Lucene’s default analyzer depends on complex, Unicode-based word boundary detection, and as such is difficult to replicate using SQL only. The same holds for LSR models that require queries to be encoded before performing retrieval. Besides the raw queries, we therefore also include pretokenized queries that have been preprocessed with Anserini’s DumpAnalyzedQueries tool, as well as cached versions of the LSR-encoded queries.⁴

Some BRIGHT collections have lists of documents to be excluded for certain queries. These lists are also included in the index lake.

2.3 Metadata Catalog

All tables are represented as Parquet files with V2 encodings, ZSTD compression, and DuckDB’s default row group size of 122,880. Each file is sorted to enable predicate pushdown and/or compression (e.g. the postings table is sorted on term to support efficient filter pushdown, and then on docid to enable gap encoding).

The tables are Hive-partitioned on columns like collection and representation (putting these values in the file path, rather than the Parquet file) to clearly separate the indexes and enable file pruning when performing retrieval over a specific index. The resulting files are all imported into a DuckLake⁵ data lake catalog. Table 1 shows the full schema of the index lake.

The index lake is distributed as a “frozen DuckLake”: all metadata is contained in a small, read-only DuckDB database file in the

same S3 bucket as the data files. This allows users to connect from DuckDB with a single ATTACH statement:

```
ATTACH 'ducklake:s3://<bucket>/anserini-indexes.ducklake';
```

Full instructions on connecting and using the index lake can be found in our Git repository.⁶

2.4 Retrieval

Retrieval can be performed through SQL, following previous work on retrieval using relational databases [4, 9]. When using the index lake, this means we download the relevant postings for the query terms and perform ranking locally. For BM25 queries, we count the number of fetched postings to compute document frequencies, and we join the postings with the lengths table:

```
SELECT docid, tf, length,
       count(*) OVER (PARTITION BY term) AS df
FROM postings NATURAL JOIN lengths
WHERE term IN ('query', 'terms')
      AND collection = '<collection>'
      AND representation = '<representation>'
      AND field = '<field>';
```

Scores are computed following Lucene’s implementation of BM25:

```
SELECT docid, sum(
    ln(1 + (num_docs - df + 0.5) / (df + 0.5)) * tf /
    (tf + k1 * (1 - b + b * (length / avg_length)))
) AS score,
FROM filtered_postings
GROUP BY docid
ORDER BY score DESC;
```

The constants num_docs and avg_length are computed on the fly, where num_docs can be fetched from the data lake’s metadata catalog and avg_length is computed from the relevant rows from the lengths table (which are downloaded anyway).

Retrieval over quantized impact scores (stored in the tf field) is more straightforward, and does not require the lengths table:

```
SELECT docid, sum(tf) AS score
FROM filtered_postings
GROUP BY docid
ORDER BY score DESC;
```

³Downloaded from <https://github.com/castorini/eval>, commit 66cc4c7.

⁴Note that new or external query sets or encoders can easily be run over the index lake as well, provided that they are tokenized similarly to the corresponding index.

⁵<https://ducklake.select/>

⁶<https://gitlab.science.ru.nl/informagus/anserini-index-lake>

Table 2: Total size (in GB) of all inverted indexes per benchmark. *Lucene* is the prebuilt index provided by Anserini; *Parquet* is the same index, converted to the Parquet-based remote index format; *DuckLake* merged all Parquet indexes into the format shown in Table 1.

Benchmark	Lucene	Parquet	DuckLake
BEIR [15]	133	35	34
BRIGHT [14]	1.6	0.59	0.56
MS MARCO [10]	307	232	197

For increased usability, the index lake also exposes macros that can be called directly. These include `tokenize` for splitting a (possibly pretokenized) query into a set of query terms and frequencies, and `score_bm25` and `score_impact` for performing retrieval over a specific index. The full list of supported macros and their arguments can be found in the Git repository.

3 Experimental Results

3.1 Index Size

First, we consider the difference in index size between Anserini’s prebuilt indexes and the index lake. Table 2 shows the total size of all included indexes per benchmark. Clearly, the indexes in the index lake are much smaller than the distributed raw Lucene indexes. For some collections, this makes sense: if the Lucene index contains a forward index alongside the inverted index, it will be much larger. However, for other collections, Anserini shares both the full index and a `slim` version, which contains the inverted index only. Since we are only performing sparse, single-stage retrieval, we only used the smaller `slim` index in such cases. For the MS MARCO collections, the DuckLake versions of the indexes are still over 40% smaller than the `slim` Lucene indexes (44 GB compared to 75 GB), confirming earlier findings that the generic tabular format of the remote index tends to be smaller than dedicated inverted index formats [2]. This is especially useful as the index needs to be fetched from remote storage, while a local system (like Lucene) might favor more lightweight compression to speed up decompression.

3.2 Effectiveness

To ensure correctness of both the converted test collections and our implementations of the relevant retrieval models, we replicated all 239 Anserini runs that used one of the prebuilt indexes also included in our index lake. We then computed the associated metrics using `trec_eval` (following the definitions stated in Anserini’s reproducibility suite) and compared them to the scores expected by Anserini. The total number of collections, topic sets and runs per benchmark is shown in Table 3.

For all impact-based retrieval models (i.e. SPLADE and uniCOIL), we obtain the exact same scores as the expected ones. This makes sense, as these models only depend on integer multiplications and are less likely to be affected by rounding differences or numerical instabilities. For our BM25 runs (including BM25F), metric scores differ by at most 0.01, likely caused by small rounding differences between Lucene’s Java implementation and DuckDB’s underlying

Table 3: Number of collections, topic sets and total number of experiments executed per benchmark.

Benchmark	# collections	# topic sets	# runs
CACM [1]	1	1	1
BEIR [15]	29	29	116
BRIGHT [14]	12	12	24
MS MARCO [10]	8	20	98
Σ	50	62	239

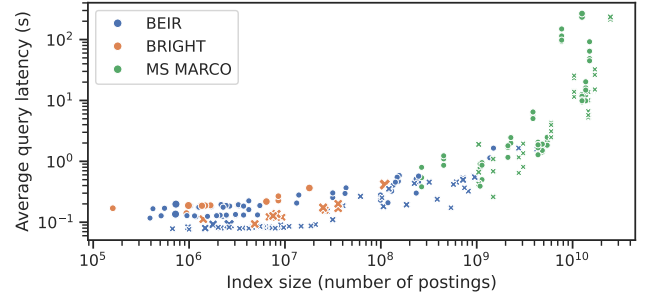


Figure 1: Average query latency per experiment, depending on the index size. Lexical indexes are represented with •, LSR indexes with ×. Marker size reflects the average query length.

C++ implementation. In our opinion, this difference is acceptable (and expected) when running the exact same queries in different languages. We conclude that Anserini’s inverted-file-based reproducibility benchmark is fully replicated in the Anserini index lake.

3.3 Query Latency

The remainder of our experimental results focus on the efficiency of running IR experiments (i.e. reproduction studies or new query sets) over the index lake. To improve the robustness of our latency measurements, we ran each query three times and report the median latency. Figure 1 shows the average latency of queries for each collection, as a function of the index size.

Over 95% of the queries can be answered within 10 seconds. Some of the queries run for much longer, though – especially when combining larger collections (like MS MARCO v2.1) with long (natural language) queries (e.g. TREC RAG). LSR models (and SPLADE-v3 in particular) can increase the query latency by a factor of almost 500, since they (expectedly) increase the size of the index and the length of the queries. In fact, some queries can take up to half an hour, due to an inefficient physical layout of the postings table. We discuss this observation in more detail in Section 4.

Note that the median latency is roughly 20% lower than the mean latency, on average. This is likely caused by two factors: (1) the first couple of queries start from a cold cache, and subsequent queries can benefit from data cached for previous queries (such as the Parquet metadata, the lengths or documents table, or postings for a term that appeared previously); and (2) most queries execute in a reasonable amount of time, but the average is influenced heavily by a few expensive queries that require many postings.

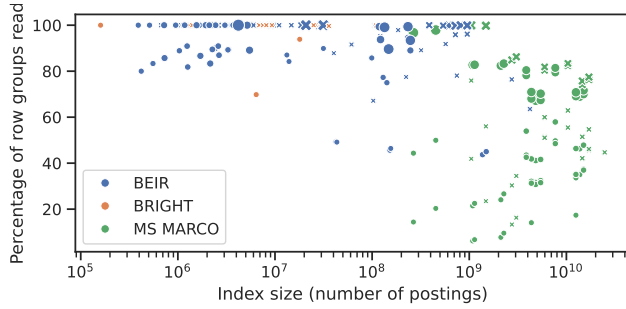


Figure 2: Index coverage (number of row groups fetched) for each experiment. Lexical indexes are represented with •, LSR indexes with ×. Marker size reflects the number of topics.

3.4 Index Coverage

While the average latency is acceptable for most queries, some of the test collections contain thousands of topics, and running the full experiment may be costly. We can instead use the remote index in a hybrid fashion by first transferring part of the tables and then running queries locally. To evaluate how well this could work, Figure 2 shows the index coverage (i.e. the total fraction of row groups fetched after running all queries) for each experiment.

For many of the smaller collections, index coverage is (close to) 100%. This makes sense, as the postings are spread out over a few row groups only, such that each row group contains a wide range of vocabulary terms and is more likely to contain any of the query terms. For the larger MS MARCO collections, index coverage drops significantly. Even the larger dev collections (which contain thousands of queries) only touch at most 80% of the index. For smaller test collections with few and/or short queries (like TREC DL), less than 10% of the index is needed.⁷

Again, we see that LSR models read more of the index than lexical retrieval, primarily due to the additional query terms introduced during query encoding. Some LSR models offer variants that only apply term (e.g. SPLADE-v3-Lexical) or skip query encoding altogether (SPLADE-v3-Doc). Such models would likely improve query latency, but have been shown to be slightly less effective.

4 Discussion

The remote index lends itself well to experiments over small query sets or collections. For larger collections (and especially LSR-based retrieval), retrieval may be relatively slow for running larger benchmarks: running a single experiment on a larger benchmark may take over 13 hours. Still, in that case, the tabular Parquet-based format easily allows researchers to download only the subset of postings they need (pushing filters on the term column into the data lake), and either import it into a local search engine or simply run the same ranking queries locally (with lower overhead).

One reason that retrieval may be costly is an inefficient physical layout of the postings table – particularly, the row group size. Large row groups can be read more efficiently because larger chunks of data are transferred at a time, while smaller row groups

can support more fine-grained filter pushdown, reducing the total amount of data read. DuckDB’s default row group size seems to be too low for our (efficiently encoded) term, docid and tf columns: the average size is only ~169 kB, resulting in severe fragmentation of the Parquet file. Each row group (or column chunk) is also fetched independently, causing thousands of tiny GET requests instead of a few larger requests that fetch multiple row groups at a time.⁸

Additionally, each index is represented using a single Parquet file, which can easily grow to dozens of gigabytes for large collections. At some point, the Parquet metadata grows significantly (especially with many small row groups), causing additional overhead for each query. Splitting such large files into multiple files could also increase efficiency – especially since DuckLake can filter out irrelevant files at the metadata layer using file-level min/max statistics.⁹

Efficiency aspects aside, our index lake mainly suffers from two limitations. For one, we cannot (easily) replicate the required tokenizers from within SQL – neither Lucene’s standard tokenizer nor the LSR encoders. However, to increase usability, it would be important to support tokenization through the index lake as well. The generic vocabulary-based tokenizer for index exchange [3] could be implemented, but unfortunately does not work with stemmed indexes. We aim to expose access to Lucene’s tokenizer and LSR encoders through a DuckDB community extension in future work.

Finally, we only considered inverted-file-based retrieval in this work, skipping the (non-negligible) subset of Anserini’s reproducibility suite related to dense retrieval. While we could share embeddings as Parquet files in remote storage, retrieval over them would be inefficient. Dense retrieval over remote and hybrid indexes is part of ongoing research, in which inverted-file-based approximate nearest neighbor search [13] seems a promising direction.

5 Conclusion

This paper documents the public Anserini *index lake* for all pre-built inverted indexes from Anserini’s reproducibility suite. We verified that the collections are represented correctly and produce correct results grounded in the reproducibility suite. While latency is acceptable for most small-to-medium-sized collections, some of the larger indexes are too large or fragmented to support retrieval experiments in reasonable time (instead running for over 13 hours) – though such experiments could still be supported by downloading the relevant set of postings and running the full experiment locally.

We aim to improve the efficiency of the index lake in future work by optimizing the physical layout of the Parquet files. We also intend to provide DuckDB bindings for the relevant tokenizers, to reduce the dependency on cached, pretokenized queries. Finally, we will consider how to integrate dense retrieval into the remote index to eventually support the entire Anserini reproducibility benchmark.

Acknowledgments

This work has received funding from the European Union’s Horizon Europe research and innovation programme under grant agreement No 101070014 (OpenWebSearch.EU, <https://doi.org/10.3030/101070014>).

⁷For example, running the TREC DL’19 topics on the msmarco-v1-doc-segmented index reads only 351 MB of Parquet data, while the full Lucene index is over 3.3 GB.

⁸Note that such behaviour is subject to change in future DuckDB releases. We also did not optimize the physical query plan of our queries – only the logical plan.

⁹In fact, DuckDB recommends keeping individual files between 100 MB and 10 GB.

References

- [1] Edward A. Fox. 1983. *Characterization of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts*. Technical Report. Cornell University.
- [2] Gijs Hendriksen, Djoerd Hiemstra, and Arjen P. de Vries. 2026. Open Web Indexes for Remote Querying. In *Advances in Information Retrieval*, Ricardo Campos, Adam Jatowt, Yanyan Lan, Mohammad Aliannejadi, Christine Bauer, Sean MacAvaney, Avishek Anand, Zhaochun Ren, Suzan Verberne, Nan Bai, and Masoud Mansoury (Eds.). Springer Nature Switzerland, Cham, 386–402. doi:10.1007/978-3-032-21289-4_25
- [3] Djoerd Hiemstra, Gijs Hendriksen, Chris Kamphuis, and Arjen P. de Vries. 2023. Challenges of Index Exchange for Search Engine Interoperability. In *Proceedings of 5th International Open Search Symposium #ossym2023*, CERN, Geneva Switzerland, 4–6 October 2023, M. Granitzer, C. Gütl, C. Plote, S. Voigt, A. Wagner (Eds.). S.I. : Open Search Foundation. doi:10.5281/zenodo.10529619
- [4] Chris Kamphuis and Arjen P. de Vries. 2021. GeeseDB: A Python Graph Engine for Exploration and Search. In *Proceedings of the Second International Conference on Design of Experimental Search & Information Retrieval Systems, Padova, Italy, September 15-18, 2021 (CEUR Workshop Proceedings, Vol. 2950)*, Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello (Eds.). CEUR-WS.org, 10–18. <https://ceur-ws.org/Vol-2950/paper-11.pdf>
- [5] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. SPLADE-v3: New Baselines for SPLADE. *CoRR* abs/2403.06789 (2024). doi:10.48550/arXiv.2403.06789
- [6] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *CoRR* abs/2106.14807 (2021). doi:10.48550/arXiv.2106.14807
- [7] Jimmy Lin, Joel Mackenzie, Chris Kamphuis, Craig Macdonald, Antonio Mallia, Michał Siedlaczek, Andrew Trotman, and Arjen de Vries. 2020. Supporting Interoperability Between Open-Source Search Engines with the Common Index File Format. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 2149–2152. doi:10.1145/3397271.3401404
- [8] Sean MacAvaney. 2025. Artifact Sharing for Information Retrieval Research. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 3974–3979. doi:10.1145/3726302.3730147
- [9] Hannes Mühleisen, Thaeer 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 International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14)*. Association for Computing Machinery, New York, NY, USA, 863–866. doi:10.1145/2600428.2609460
- [10] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches 2016 Co-Located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016 (CEUR Workshop Proceedings, Vol. 1773)*, Tarek Richard Besold, Antoine Bordes, Artur S. d'Ávila Garcez, and Greg Wayne (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf
- [11] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1981–1984. doi:10.1145/3299869.3320212
- [12] Harrison Scells, Shengyao Zhuang, and Guido Zuccon. 2022. Reduce, Reuse, Recycle: Green Information Retrieval Research. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2825–2837. doi:10.1145/3477495.3531766
- [13] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*. 1470–1477 vol.2. doi:10.1109/ICCV.2003.1238663
- [14] Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Liu Haisu, Quan Shi, Zachary Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Serkan Arik, Danqi Chen, and Tao Yu. 2025. BRIGHT: A Realistic and Challenging Benchmark for Reasoning-Intensive Retrieval. In *International Conference on Learning Representations*, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (Eds.), Vol. 2025. 48941–48991. https://proceedings.iclr.cc/paper_files/paper/2025/file/7a0f8055c838df8e62329a76c7c6403d-Paper-Conference.pdf
- [15] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/65b9eea6e1cc6bb9f0cd2a47751a186f-Paper-round2.pdf
- [16] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 1253–1256. doi:10.1145/3077136.3080721