Evaluating the Efficiency and Effectiveness of Learned Sparse Retrieval with the `lsr_benchmark`

Maik Fröbe[1], Ferdinand Schlatt[1], Cosimo Rulli[2], Tim Hagen[3], Jan Heinrich Merker[1], Gijs Hendriksen[4], Carlos Lassance[5], Franco Maria Nardini[2], Rossano Venturini[6], and Martin Potthast[7]

[1] Friedrich-Schiller-Universität Jena
[2] ISTI-CNR
[3] University of Kassel and hessian.AI
[4] Radboud University
[5] Cohere
[6] University of Pisa
[7] University of Kassel, hessian.AI, and ScaDS.AI

Abstract. Learned sparse retrieval (LSR) models exhibit varying trade-offs between effectiveness and efficiency. But while standard tools exist for evaluating LSR effectiveness, there is none for evaluating efficiency. Also, datasets with high-quality relevance judgments are too large for repeated efficiency experiments, e.g., on different hardware configurations. To promote the evaluation of LSR models in terms of their effectiveness and efficiency, we introduce the `lsr_benchmark`, which measures retrieval efficiency at each step of an LSR pipeline (document embedding, indexing, query embedding, and retrieval) as well as its overall effectiveness. To ensure tractability and extensibility, we apply current corpus subsampling methods to eleven TREC tasks, precompute embeddings with eleven LSR models per task, and evaluate eight retrieval engines as baselines. For the benchmark's hosted version, a modular API, along with tools for evaluating effectiveness and efficiency, facilitates the submission of new approaches. Our experiments show that the chosen embedding model significantly affects the efficiency of a retrieval engine and that LSR is more effective but less efficient than BM25—an efficiency gap that our benchmark now tracks as new LSR models are published.

Keywords: Learned Sparse Retrieval · Neural IR · Green IR Evaluation

1 Introduction

Learned sparse retrieval (LSR) models embed documents and queries into sparse embeddings that weight terms according to their importance in a document or query [46]. They use transformer-based language models to derive these term weights, in contrast to lexical models such as BM25 [49], which use corpus statistics. In this way, LSR integrates what is known in lexical retrieval as query or document expansion/reduction with embedding into a sparse latent term space, increasing effectiveness compared to lexical retrieval [26, 27, 35, 37].

Table 1: The datasets in the `lsr_benchmark` together with the embeddings that we release publicly for retrieval experiments without access to the corpus.

Corpus	Tracks	Queries	Judgm.	Docs.	Embeddings	
					Avg.	$\sum$
ClueWeb09	Web [13–16]	198	84 366	315 095	117.9 MB	5.1 GB
ClueWeb12	Web/Dec. [1, 17, 18]	150	51 765	225 636	116.0 MB	3.7 GB
Disks 4/5	Robust04 [59]	249	311 410	285 756	106.4 MB	5.7 GB
MS MARCO	DL 19/20 [20, 21]	97	20 646	81 737	40.1 MB	0.9 GB
MS $\text{MARCO}_{2.1}$	RAG 24 [57]	89	20 429	116 694	170.0 MB	1.8 GB
$\sum$	11 TREC Tracks	783	488 616	1 024 918	106.8 MB	17.2 GB

While the term weights differ between lexical and learned sparse retrieval [41], the representations of queries and documents as high-dimensional sparse vectors are structurally identical. Therefore, retrieval systems that process lexical representations [39, 44] can also process LSR representations. But the different term weight distributions cause efficiency optimizations that used to work for lexical retrieval [3, 56] to fail for LSR [41], so that new optimizations are required [7, 22].

Retrieval engines that have been optimized for learned sparse retrieval substantially improved the latency for LSR embeddings [7, 22]. However, previous efficiency experiments did not compare retrieval engines across diverse retrieval scenarios and many LSR models, as many IR evaluations build on web-scale datasets that cannot be processed with LSR models with reasonable cost-effectiveness due the high embedding costs. To enable holistic learned sparse retrieval evaluations that account for effectiveness and efficiency, we present the `lsr_benchmark`. We assume a standard four-step learned sparse retrieval pipeline where (1) documents are embedded and (2) indexed, after which (3) queries are embedded to (4) retrieve results. Efficiency is monitored at each step with the `tirex_tracker` [33] and we persist all measurements in the `ir_metadata` format [2] to make runs fully self-contained. We decouple the embedding step from the retrieval step to ensure that all combinations of embedding models and retrieval engines can be explored easily. We use the recently proposed corpus subsampling methods by Fröbe et al. [29] to enable, for the first time, a systematic evaluation of complex learned sparse retrieval models as first-stage retrievers on web-scale corpora. Table 1 provides an overview of the datasets that we include into the first version of the `lsr_benchmark`.

Our experiments show that there is still a substantial gap between lexical and learned sparse retrieval. LSR achieves higher effectiveness, at the cost of latency in the 90 % percentile, where retrieval latency increases 24-fold across retrieval engines and LSR embeddings. The framework, submission instructions, and aggregated efficiency and effectiveness evaluations are available online.[8]

[8] Code, tutorials, and documentation: https://github.com/reneuir/lsr-benchmark
Submission and evaluations: https://www.tira.io/task-overview/lsr-benchmark

2 Related Work

We review learned sparse retrieval, corpus subsampling, green and efficient IR, and on tools which the `lsr_benchmark` aims to support and builds upon.

Learned Sparse Retrieval (LSR). Early retrieval systems used lexical priors (e.g., term frequency and inverse document frequency [54]) to represent documents. Such representations are sparse (the number of unique terms in a document is orders of magnitude smaller than the number of possible terms) but can struggle to capture semantics. Learned sparse retrieval improves semantic matching by replacing lexical priors with priors from deep learning models [12, 27, 37, 62, 46].

Since lexical and LSR models both use structurally identical output representations, retrieval engines that can handle lexical retrieval can also handle learned sparse retrieval. However, efficient lexical retrieval engines exhibit a tenfold increase in latency when applied to learned sparse representations [41]. Later works improved LSR-latency by specifically designed retrieval engines [7, 8, 40, 42, 47]. However, LSR efficiency is often only evaluated on few datasets and models. As our experiments include many retrieval scenarios, embedding models, and retrieval algorithms, we can reproduce cases with previously reported latency differences, but we also find cases where the latency gap between lexical and LSR increases 70-fold, emphasizing the need for further investigations.

Corpus Subsampling. High quality evaluation corpora constructed in TREC-style shared tasks come with many graded relevance judgments (beneficial for nDCG) and their reliability for subsequent evaluations can be tested [58]. However, TREC-style collections are often very large (even web-scale), as TREC aimed to transfer the Cranfield Paradigm to large document collections [60]. The size of TREC collections causes problems for the evaluation of modern neural models that have high energy/compute requirements [51]. While building a lexical ClueWeb09 index takes less than a day with PISA [44], embedding 50 % of the ClueWeb09 on an Nvidia V100 GPU takes 71 days [36]. Those high computational costs make it impossible to run efficiency-oriented experiments with neural models and multiple repetitions on web-scale corpora. Corpus subsampling [29] reduces the size of document collections so that expensive neural models can also be reliably evaluated on large corpora. All runs that were pooled are used to take the top-k documents of all pooled systems, where k is substantially larger than the pooling depth, which yields diverse hard-negative documents. We use this to run experiments on collections that would otherwise not be possible.

Green IR and Efficiency. Several studies investigate how to optimize search engine efficiency from the perspective of energy consumption. Catena et al. [11] propose specialized CPU governors that use information on the query server's load to adjust the CPU frequency to reduce the energy consumption which later evolved into energy-aware schedulers that can reduce CPU energy consumption by up to 50 % [10]. With the wide use of large language models, numerous contributions have emphasized that their improved effectiveness comes at the cost

of substantially higher hardware and energy demands. Several studies address this issue through strategies aimed at mitigating such resource drifts [51, 55]. In this direction, the ReNeuIR workshop at SIGIR fostered an active discussion on alleviating the computational burden of these models [4–6, 28]. This ongoing work highlights the need for a holistic and concrete definition of efficiency, as well as the existence of several open research gaps in efficiency-centered evaluation. To help to bridge these gaps, the ReNeuIR community has initiated a shared task initiative. We aim to support such initiatives with our work.

Related Tooling for Simplified IR Experiments. The ongoing discussion on how to conduct efficiency-oriented IR experiments [4, 6, 28] that inspired our work and to which we aim to contribute, needs tool support. Therefore, we build our `lsr_benchmark` compatible with existing IR tooling. The `ir_datasets` [38] framework provides an API for accessing IR evaluation datasets. Our API is compatible with `ir_datasets` and extends it by adding sparse embeddings. Additionally, we need to enable efficiency evaluations. For this, we incorporate the `tirex_tracker` [33], a lightweight native library that can be embedded into many programming languages and monitors all efficiency metrics, thereby following the spirit that there is not yet a consensus on how to measure efficiency. Compared to alternatives such as codecarbon [19], `tirex_tracker` is independent of the programming language, captures the efficiency over time, and can export the monitored resource consumption in the `ir_metadata` format [2], yielding higher compatibility with IR tools. Our `lsr_benchmark` is the first project that uses the `tirex_tracker` in efficiency-oriented experiments to collect efficiency measurements of IR workloads at scale. We aim to be orthogonal and to support theoretically oriented efficiency frameworks such as PEIR [34] (which uses theoretical modeling to assess efficiency). We see our work as an initiative in this direction and hope to inspire similar initiatives in the IR community.

3 Overview of the `lsr_benchmark`

The `lsr_benchmark` aims to support holistic evaluations of Learned Sparse Retrieval (LSR) methods, accounting for efficiency and effectiveness. Our LSR pipelines first embed documents and queries into a standardized format (Section 3.2), and then perform the index and retrieve step (Section 3.3). We use TIRA/TIREx [30, 31] for the embedding as it allows to process datasets that are not public. The subsequent retrieval experiments can run with any frameworks and infrastructures, while all steps are monitored with the `tirex_tracker` [33] to capture efficiency-oriented metrics in the `ir_metadata` [2] format. We designed the architecture of the `lsr_benchmark` (Section 3.1) to have a low barrier of entry.

The `lsr_benchmark` is pip-installable. Listing 1 shows how public embeddings and runs can be accessed via the command line. The public embeddings are used as input for retrieval engines, without needing the underlying corpus (only the embeddings are public). The public runs serve as baselines and for exploratory analysis. Listing 2 shows the `evaluate` command that uses run files as input

```
# download pre-computed embeddings
lsr-benchmark download-embeddings \
    --embedding webis/splade \
    --dataset msmarco-passage/trec-dl-2019

# download a run
lsr-benchmark download-run \
    --embedding webis/splade \
    --dataset msmarco-passage/trec-dl-2020 \
    --retrieval seismic
```

Listing 1: The command line interface of the `lsr_benchmark` for downloading public pre-computed embeddings (allows easy experiments) and baseline runs.

```
lsr-benchmark evaluate OUTPUT-BY-SEISMIC OUTPUT-BY-NAIVE-SEARCH
```

	Seismic	**Naïve-Search**
index.runtime_wallclock	32365 ms	0 ms
index.energy_total	7.0	0.0
retrieval.runtime_wallclock	35 ms	858 ms
retrieval.energy_total	0.0	0.0
embedding/model	webis/splade	webis/splade
embedding/doc.runtime_wallclock	96789 ms	96789 ms
embedding/doc.energy_total	17896.0	17896.0
embedding/query.runtime_wallclock	1575 ms	1575 ms
embedding/query.energy_total	109.0	109.0
nDCG@10	0.720	0.720
ir_dataset	msmarco-passage/trec-dl-2020	

Listing 2: The `evaluate` command of the `lsr_benchmark` on the command line for comparing runs in terms of efficency and effectiveness and the output.

and outputs effectiveness and efficiency measures, e.g., the energy required to embed the documents and queries. The example (Listing 2) compares Seismic [7] with Naïve Search (a simple linear scan) on the same embeddings, showing that Seismic is optimized for low-latency retrieval (`retrieval.runtime_wallclock` of 35 *msec.* vs 858 *msec.*) while it has substantially higher indexing costs.

3.1 Architecture of the `lsr_benchmark`

We operationalize the `lsr_benchmark` in four stages that (1) add new datasets, (2) embed queries and documents, (3) run retrieval engines, and (4) compare efficiency and effectiveness. The first two stages run in the TIRA sandbox (open for submissions), and the last two stages operate on the artifacts published from the embedding stage, so that all retrieval experimentation is completely open and can be executed on the side of the experimenter with the tools of their choice. Figure 1 overviews this architecture that we explain next.

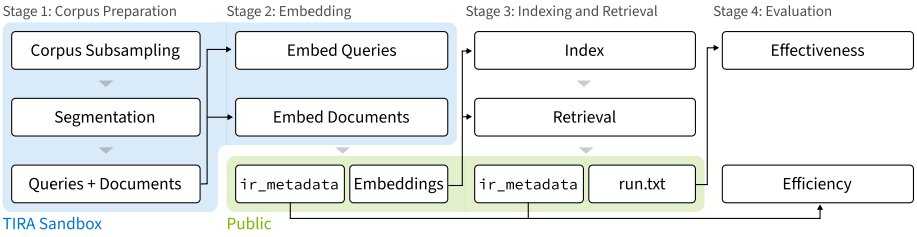

Fig. 1: The architecture of the `lsr_benchmark`. Stage one and two run in TIRA to process non-public datasets. Stages three and four build on the embeddings published in stage two and can run on any infrastructure of an experimenter.

Stage 1 (Adding Corpora). The process starts by uploading a new dataset to TIRA. We release a detailed guide together with code examples in the documentation.[9] We focus on TREC-style datasets that have many high-quality relevance judgments from pooling, as this allows us to assess the reliability of evaluations [58]. To add a new dataset, we download all runs submitted to the corresponding TREC tracks to run corpus subsampling [29]. For each dataset, we test for subsampling depths of 100 (was already reliable [29]), 125, 150, and 200 to select a configuration that yields a reasonable corpus size. Note that the corpus subsampling that we use achieved the best reliability compared to other sampling strategies, and that this subsampling is required, as otherwise no experimentation would be possible. After the subsampling, we segment the documents into passages so that embedding approaches that aggregate multiple passages use the same segmentation. We then upload the subsampled and segmented corpora to TIRA. For public corpora (e.g., MS MARCO), the subsamples are public (to allow for local experimentation) while we configure that datasets that can not be publicly shared (e.g., Robust04 and the ClueWebs) are only available in TIRA.

Stage 2 (Embedding Documents and Queries). Encoders take the queries and the documents as text and embed them into sparse vectors, commonly represented as a list of term–score tuples. We submit the encoders to TIRA, where they are used to produce the embeddings in the TIRA sandbox without network access—improving reproducibility and ensuring that the data remains private. We also ensure that all embedding models run on the same host, and we track the resource consumption during encoding with the `tirex_tracker`.

Stage 3 (Indexing and Retrieval). Retrieval engines use query and document embeddings to (1) build an index, and (2) perform retrieval. Both stages are monitored with the `tirex_tracker`. We build and publish a set of 8 retrieval engines that can be used as baselines on any target architectures.

Stage 4 (Efficiency and Effectiveness Evaluations). After Stage 3 yields retrieval runs, evaluating efficiency and effectiveness as in Listing 2 are possible.

[9] https://github.com/reneuir/lsr-benchmark/tree/main/data

```python
with tracking(export_file_path=output_dir / "query"):
    query_embeddings = model.predict(query_texts)

with tracking(export_file_path=output_dir / "docs"):
    doc_embeddings = model.predict(texts)

save_embeddings(query_embeddings, doc_embeddings, output_dir)
```

Listing 3: Example of how queries and documents are embedded.

```python
for query_id, tokens, values in query_embeddings:
    print(query_id) # '1030303'
    print(tokens)   # ['2002' '2010' '2032' ... ]
    print(values)   # [0.02466838 0.49301645 0.99693686 ... ]
```

Listing 4: Queries/documents are embedded as token ids and importance scores.

As the `tirex_tracker` automatically captures all efficiency information in the `ir_metadata` format, different aspects of efficiency can be evaluated in retrospect.

3.2 Embeddings for Learned Sparse Retrieval

Listing 3 shows how we track the efficiency of embedding documents and queries. Loading models and persisting the results is done outside the tracking. We use Lightning IR [52] to run the inference for most models, as it offers a unified API for different model types (other frameworks can be integrated). We include 11 different models in our comparison;[10] mostly SPLADE variants [27] (covering smaller backbones [50] and inference-free architectures for the query [53, 32, 45]) as well as UniCoil [37] and the sparse variant of the BGE-M3 model [61].

Listing 4 illustrates how queries and documents are represented and how they can be accessed. Each query and document is represented by a list of token ids and corresponding embedding values. Note that we can publish these embeddings, even for restrictively licensed datasets, as one cannot reproduce the original text from the embeddings without substantial effort.

3.3 Retrieval Engines for Learned Sparse Retrieval

The first version of our `lsr_benchmark` comes with eight retrieval engines. All integrations monitor the efficiency of indexing and retrieval with a protocol as showcased in Listing 5 to load, index, and evaluate any of the embeddings.

DuckDB [48] is a production ready and efficiency-oriented database system that also comes with tooling to support learned sparse retrieval.

Naïve Search represents a simple baseline that exhaustively computes dot products without indexing (implemented in Rust in the SEISMIC package).

[10] bge-m3 OS2 Dist OS2 Doc OS2 Mini OS3 Dist Splade Splade-v2 Dist Splade-v3 Splade-v3 Dist Splade-v3 Doc UniCoil

```
lsr_benchmark.register_to_ir_datasets("<DATASET-ID>")
ir_dataset = ir_datasets.load("lsr-benchmark/<DATASET-ID>")
doc_embeddings = ir_dataset.doc_embeddings(model_name="<MODEL>")
query_embeddings = ir_dataset.query_embeddings(model_name="<MODEL>")

with tracking(export_file_path=output_dir / "index"):
    index = build_index(document_embeddings)

with tracking(export_file_path=output_dir / "retrieval"):
    run = retrieval(index, query_embeddings)

save_run(output_dir, output_dir)
```

Listing 5: Indexing and retrieval while capturing efficiency metrics.

PISA [44] is a C++ library for efficiency-oriented research, originally for lexical representations that comes with several pruning algorithms [3, 23, 43, 56]. (We include two variants of PISA, one for lexical and one for sparse retrieval.)

PyTerrier [39] is a research-oriented retrieval engine that is mostly written in Java and aims to scale to large datasets with declarative experimentation. (We include two variants of PyTerrier, one for lexical and one for sparse retrieval.)

kANNolo [22] is a Rust framework for approximate nearest neighbors search for dense and sparse retrieval, focusing on graph-based algorithms (e.g., HNSW).

Seismic [7–9] implements approximate nearest neighbor search in Rust for sparse embeddings with block partitioning, forward indices, and skip vectors.

3.4 Submission of new Embedding Models and Retrieval Engines

We encourage the community to submit new embedding models and new retrieval engines to the `lsr_benchmark`. We envision that efficiency-oriented shared tasks and workshops, such as ReNeuIR [6, 4, 28] and WOWS [25, 24][11] incorporate aspects of the `lsr_benchmark` to grow a community around it. We organize the `lsr_benchmark` in a single mono-repository that contains all code in a clean and consistent structure. Hence, we intend to collect new submissions via pull requests. All code for embedding models and retrieval engines is "dockerized" and compatible with Development Containers,[12] reducing the effort to deploy them on new hardware. For submitting new models, our Lightning IR code should, in many cases, allow TIRA submissions from the existing code with just linking against a different/new Hugging Face model. For collecting diverse efficiency and effectiveness information for retrieval engines, we allow for uploading runs to TIRA, so that executions of the retrieval engines that we dockerized on diverse infrastructure yield a valuable parallel dataset (runs with their monitored executions) with effectiveness and efficiency information. This setup ensures that no prior experience with TIRA is needed to participate in the `lsr_benchmark`.

[11] https://reneuir.org/ https://opensearchfoundation.org/wows2025/
[12] https://containers.dev/

Table 2: The efficiency of six LSR and two lexical retrieval engines measured as wallclock runtime in milliseconds (50 %, 90 %, and 99 % percentiles) and consumed energy in Joules (average and total) for indexing (runtime per 1 000 documents) and retrieval (runtime per query) accross all embeddings and collections.

Engine	Runtime (ms)						Energy (J)			
	Index			Retrieval			Index		Retrieval	
	50	90	99	50	90	99	Avg.	Tot.	Avg.	Tot.
DuckDB	31.9	54.3	87.3	12.1	113.6	191.0	0.3	4.2	0.2	34.0
kANNolo	812.6	1206.8	1493.5	1.1	1.9	2.3	13.3	199.5	0.0	0.0
Naïve	0.0	0.0	0.0	10.2	55.5	73.7	0.0	0.0	0.0	7.0
PISA	162.6	219.0	281.2	4.3	31.5	76.1	2.3	34.1	0.0	4.0
PyTerrier	355.8	500.2	685.5	25.5	53.6	90.0	5.5	81.9	0.0	8.0
Seismic	1276.8	2289.3	3366.1	0.7	3.0	4.4	21.7	325.3	0.0	0.0
BM25@PyTerrier	789.2	7886.8	16012.6	16.4	18.4	19.2	69.6	1044.0	0.0	0.0
BM25@PISA	252.3	419.2	456.8	0.6	0.8	1.3	4.9	73.0	0.0	0.0

4 Evaluation

We show use cases and future perspectives of the `lsr_benchmark` and report efficiency/effectiveness on our 11 datasets, 11 embedding models, and 8 retrieval engines (968 runs). A leaderboard (hopefully growing over time) is available.[13]

4.1 The Efficiency of Learned Sparse Retrieval

We first analyze how retrieval engines and learned sparse embedding models impact the efficiency. We focus our efficiency evaluations on latency and energy consumption of the embedding, indexing, and retrieval captured with the `tirex_tracker` (other efficiency metrics such as CPU utilization, RAM usage, etc. are captured in the `ir_metadata`, but we do not include them in the paper). All embedding models were executed in TIRA on the same machine with Nvidia A100 GPUs, and all retrieval experiments were executed on another machine outside of TIRA (hardware specifications are included in the runs' `ir_metadata`).

Table 2 provides an overview of the efficiency of the eight retrieval engines for indexing and retrieval. The energy is reported on average (i.e., per dataset and embedding) and total (for processing all datasets and embeddings). For latency, we report the 50 %, 90 %, and 99 % percentiles and normalize the values such that they report the elapsed time to index 1000 documents and the time to run retrieval for one query (the energy and latency of embedding documents and queries are excluded for now as we focus on this later). We observe that the retrieval engine has a substantial impact on the efficiency. Engines that allow for low-latency retrieval invest substantial effort into the indexing stage, with Seismic having the highest indexing latency (1.3 seconds to index 1000 documents in the 50 % percentile) that yields the best retrieval latency (0.7 milliseconds in the 50 % percentile). DuckDB and PyTerrier have at many percentiles a higher

[13] https://www.tira.io/task-overview/lsr-benchmark

Table 3: Efficiency/Effectiveness comparison of retrieval engines for learned sparse retrieval versus lexical BM25 retrieval in latency in milliseconds at 50 %, 90 %, and 99 % percentiles and the corresponding nDCG@10 effectiveness.

Engine	**Lerned Sparse**				**Lexical (BM25)**			
	Retrieval (ms)			**nDCG@10**	**Retrieval (ms)**			**nDCG@10**
	50	90	99		50	90	99	
DuckDB	12.35	108.56	188.71	0.385	6.75	8.34	8.51	0.266
Naïve	9.57	54.82	73.51	0.385	3.30	5.26	7.57	0.266
PISA	3.27	31.34	69.46	0.385	0.76	1.06	1.57	0.266
PyTerrier	24.79	51.59	89.53	0.385	20.25	26.32	26.81	0.266
Seismic	0.70	2.94	4.30	0.38	0.04	0.04	0.05	**0.216**

retrieval latency than the naïve search, which highlights that low-latency is not the only requirement for production-ready systems. Lexical retrieval (the last two rows) is less prone to latency problems than learned sparse retrieval (the 90 % and 99 % percentiles are very close to the 50 % percentile).

To repeat the study if lexical retrieval allows for more efficient retrieval than learned sparse retrieval (observed in prior work [41] and in Table 2), we build BM25 embeddings for every dataset (not counted in the 11 embeddings). These BM25 embeddings map every term in the query to an importance of 1, and every document term to its BM25 score that we calculate with PyTerrier. In contrast to learned sparse embeddings, the vocabulary size is substantially larger (yielding shorter posting lists). We run all retrieval engines on the BM25 embeddings and compare the efficiency of lexical retrieval with learned sparse retrieval. Only PISA and PyTerrier are initially built for lexical retrieval, whereas engines such as kANNolo and Seismic focus on learned sparse retrieval. The kANNolo retrieval engine failed to process BM25 embeddings (the maximum vocabulary size of kANNolo is 2^{16}, which is too small for the lexical vocabulary of our datasets). For all other retrieval engines, we show the difference in the retrieval latency and effectiveness for learned sparse retrieval (across all 11 embedding models) and lexical BM25 retrieval in Table 3. Across all reported percentiles and all retrieval engines, BM25 retrieval is faster than learned sparse retrieval. Especially in the higher percentiles, the efficiency difference is substantial, for instance, the average speedup in the 90 % percentile retrieval latency is 24× across the retrieval engines (between 10× for Naïve and 70× for Seismic). Seismic is even faster than PISA in lexical retrieval, but this comes at effectiveness reductions. The nDCG@10 for Seismic is lower for lexical retrieval than for the other engines, as Seismic prunes the search space optimized for learned sparse retrieval, while the other engines do exact search. This highlights that different retrieval engines can have a substantial impact on retrieval efficiency but also that there can be an impact on effectiveness that we study next.

Table 4: The nDCG@10 and P@10 of eleven LSR models and two lexical BM25 baselines on all test collections with the energy in Joules for the embeddings.

Embedding		**nDCG@10**						**P@10**					
Model	Energy	CW09	CW12	DL	R04	RAG	Avg.	C09	C12	DL	R04	RAG	Avg.
BGE-M3	509815	.09	.23	.40	.34	.20	.25	.14	.30	.47	.31	.30	.28
OS2 Dist	45427	.24	.34	.72	.49	.43	.42	.33	.43	.80	.47	.59	.48
OS2 Doc	45345	.23	.33	.69	.46	.37	.40	.32	.41	.76	.44	.53	.45
OS2 Mini	31691	.23	.33	.69	.44	.35	.39	.30	.40	.75	.42	.51	.44
OS3 Dist	46885	.24	.33	.70	.46	.37	.40	.32	.40	.77	.44	.52	.45
Splade	42445	.22	.32	.74	.47	.42	.40	.31	.40	.81	.45	.56	.46
Splade-v2 Dist	45490	.21	.33	.72	.48	.37	.40	.29	.42	.80	.46	.52	.46
Splade-v3	68183	.22	.34	.74	.49	.44	.42	.31	.42	.82	.47	.59	.47
Splade-v3 Dist	45543	.23	.33	.75	.48	.39	.41	.31	.43	.82	.46	.55	.47
Splade-v3 Doc	68326	.16	.28	.71	.43	.32	.36	.22	.36	.77	.42	.47	.41
UniCoil	54712	.18	.27	.61	.38	.32	.33	.24	.34	.68	.38	.45	.38
BM25 PyTerrier		.11	.30	.48	.40	.26	.30	.16	.40	.57	.38	.36	.35
BM25 PISA		.10	.28	.48	.41	.25	.30	.14	.37	.57	.39	.35	.34

4.2 The Effectiveness of Learned Sparse Retrieval

We run all 8 retrieval engines on all 11 LSR models to study their effectiveness. For learned sparse retrieval, we observe that the retrieval engine has no impact on the effectiveness (nDCG@10 scores differ only in the third decimal place). Table 4 reports the energy needed to build the embeddings and their effectiveness in nDCG@10 and Precision@10 for all corpora (macro averaged), together with the effectiveness of BM25 in PISA and PyTerrier (they use different defaults). Most LSR engines are more effective than BM25 (UniCoil and BGE-M3 being rather ineffective) and Splade v3 and OS2 Dist. achieving the highest nDCG@10 (we run no significance tests as our analysis is explorative).

We analyze the impact of different LSR models for all 8 retrieval engines on the indexing and retrieval latency in Table 5. Different LSR models produce different term distributions, and we observe that highly effective models are challenging for efficient indexing and retrieval. UniCoil needs in the 50 % percentile 3.2 milliseconds per query whereas highly effective systems like Splade v3 and OS2 Dist. need 10.3 respectively 32.0 milliseconds. The 99 % percentiles contain cases where retrieval engines are highly inefficient (reaching 100 milliseconds).

4.3 Perspectives

With the `lsr_benchmark`, we support efficiency and effectiveness evaluations to enable new research perspectives. We intend to support shared tasks and hope that also other directions are enabled that are interesting to the community.

Continuous Intergration. The `lsr_benchmark` has a focus to support the development of efficient retrieval, but needs to process ca. 20 GB of embeddings. Deriving

Table 5: The latency for indexing and retrieving LSR representations of different LSR models in milliseconds for all retrieval engines at 50 %, 90 %, and 99 %.

Model	Index			Retrieval		
	50	90	99	50	90	99
BGE-M3	62.1	671.0	1621.5	5.5	25.5	29.7
OS2 Dist.	221.3	1053.8	1562.5	32.0	104.8	164.7
OS2 Doc	203.3	1078.8	1985.8	4.4	21.4	27.0
OS2 Mini	270.0	1305.0	2741.7	4.4	20.9	26.0
OS3 Dist.	198.3	937.8	1276.8	4.5	20.9	24.4
Splade	227.4	1331.9	2021.0	37.7	129.3	195.4
Splade2 Dist.	291.7	1762.9	3475.3	21.0	62.9	95.2
Splade3	220.6	1268.6	2170.7	10.3	39.3	57.0
Splade3 Dist.	243.3	1443.5	2321.5	9.6	43.3	69.0
Splade3 Doc	225.7	1020.3	1949.1	4.6	21.1	27.4
UniCoil	100.7	695.2	860.0	3.2	27.6	35.4

micro-benchmarks, e.g., via stratifying efficiency percentiles, could yield evaluations that can run on every commit. Developing micro benchmarks that run fast and correlate with the full `lsr_benchmark` would enable continuous intergration.

Interpolation Between Lexical and Learned Sparse Retrieval. LSR and lexical retrieval already share the same underlying representations. One interesting direction would be that a retrieval engine can, depending on the query, interpolate between lexical and learned sparse retrieval (e.g., via efficiency predictions).

Red-Teaming for Efficiency Evaluations. Given our focus on efficiency, it would be interesting to include retrieval engines that "purposefully cheat" (e.g., efficiency oracles). Such oracles are not reachable but help to identify errors or room for improvement (e.g., knowing which queries come during retrieval).

Enrichment of the Corpora. That we publish the document and query embeddings allows for many additional resources that could complement our work. For instance, building query variants only needs access to the public topics and allows to study the efficiency of LSR for queries of different verbosity.

5 Conclusion

We presented the `lsr_benchmark` for holistic evaluations of learned sparse retrieval of efficiency and effectiveness. We supportrivate datasets derived from web-scale corpora via corpus subsampling by running the LSR models within TIRA and releasing only the resulting embeddings publically so that retrieval engines can directly work on prepared embeddings. We showcased several use-cases for efficiency and effectiveness-oriented evaluations, highlighting that the LSR model substantially impacts the efficiency of an retrieval engine. Our released resources allow to systematically identify efficiency problems at a low barrier of entry to help addressing those problems in a sustainable and reproducible way.

References

[1] Abualsaud, M., Lioma, C., Maistro, M., Smucker, M.D., Zuccon, G.: Overview of the TREC 2019 decision track. In: Proc. of TREC 2019 (2019)

[2] Breuer, T., Keller, J., Schaer, P.: ir_metadata: An extensible metadata schema for IR experiments. In: Proc. of SIGIR 2022, pp. 3078–3089, ACM (2022)

[3] Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., Zien, J.: Efficient query evaluation using a two-level retrieval process. In: Proc. of CIKM 2003, pp. 426–434 (2003)

[4] Bruch, S., Fröbe, M., Hagen, T., Nardini, F.M., Potthast, M.: Reneuir at SIGIR 2025: The fourth workshop on reaching efficiency in neural information retrieval. In: Proc. of SIGIR 2025, pp. 4153–4156, ACM (2025)

[5] Bruch, S., Lucchese, C., Nardini, F.M.: Reneuir: Reaching efficiency in neural information retrieval. In: Proc. of SIGIR 2022, p. 3462–3465, ACM (2022)

[6] Bruch, S., Mackenzie, J., Maistro, M., Nardini, F.M.: Reneuir at SIGIR 2023: The second workshop on reaching efficiency in neural information retrieval. In: Proc. of SIGIR 2023, pp. 3456–3459, ACM (2023)

[7] Bruch, S., Nardini, F.M., Rulli, C., Venturini, R.: Efficient inverted indexes for approximate retrieval over learned sparse representations. In: Proc. of SIGIR 2024, pp. 152–162 (2024)

[8] Bruch, S., Nardini, F.M., Rulli, C., Venturini, R.: Pairing clustered inverted indexes with knn graphs for fast approximate retrieval over learned sparse representations. In: Proc. of CIKM 2024 (2024)

[9] Bruch, S., Nardini, F.M., Rulli, C., Venturini, R., Venuta, L.: Investigating the scalability of approximate sparse retrieval algorithms to massive datasets. In: Proc. of ECIR 2025, pp. 437–445 (2025)

[10] Catena, M., Frieder, O., Tonellotto, N.: Efficient energy management in distributed web search. In: Proc. of CIKM 2018, pp. 1555–1558, ACM (2018)

[11] Catena, M., Macdonald, C., Tonellotto, N.: Load-sensitive CPU power management for web search engines. In: Proc. of SIGIR 2015, pp. 751–754, ACM (2015)

[12] Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., Liu, Z.: Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv preprint arXiv:2402.03216 (2024)

[13] Clarke, C.L.A., Craswell, N., Soboroff, I.: Overview of the TREC 2009 Web track. In: Proc. of TREC 2009, NIST (2009)

[14] Clarke, C.L.A., Craswell, N., Soboroff, I., Cormack, G.V.: Overview of the TREC 2010 Web track. In: Proc. of TREC 2010, NIST (2010)

[15] Clarke, C.L.A., Craswell, N., Soboroff, I., Voorhees, E.M.: Overview of the TREC 2011 Web track. In: Proc. of TREC 2011, NIST (2011)

[16] Clarke, C.L.A., Craswell, N., Voorhees, E.M.: Overview of the TREC 2012 Web track. In: Proc. of TREC 2012, NIST (2012)

[17] Collins-Thompson, K., Bennett, P.N., Diaz, F., Clarke, C., Voorhees, E.M.: TREC 2013 Web track overview. In: Proc. of TREC 2013, NIST (2013)
[18] Collins-Thompson, K., Macdonald, C., Bennett, P.N., Diaz, F., Voorhees, E.M.: TREC 2014 Web track overview. In: Proc. of TREC 2014, NIST (2014)
[19] Courty, B.: mlco2/codecarbon: v2.4.1 (May 2024), URL https://doi.org/10.5281/zenodo.11171501
[20] Craswell, N., Mitra, B., Yilmaz, E., Campos, D.: Overview of the TREC 2020 Deep Learning Track. In: Proc. of TREC 2020, NIST (2020)
[21] Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M.: Overview of the TREC 2019 Deep Learning Track. In: Proc. of TREC 2019, NIST (2019)
[22] Delfino, L., Erriquez, D., Martinico, S., Nardini, F.M., Rulli, C., Venturini, R.: kannolo: Sweet and smooth approximate k-nearest neighbors search. In: Proc. of ECIR 2025, pp. 400–406 (2025)
[23] Ding, S., Suel, T.: Faster top-k document retrieval using block-max indexes. In: Proc. of SIGIR 2011, pp. 993–1002, ACM (2011)
[24] Farzana, S.M., Fröbe, M., Granitzer, M., Hendriksen, G., Hiemstra, D., Potthast, M., de Vries, A.P., Zerhoudi, S.: Report on the 1st international workshop on open web search (WOWS 2024) at ECIR 2024. SIGIR Forum **58**(1), 1–13 (2024)
[25] Farzana, S.M., Fröbe, M., Granitzer, M., Hendriksen, G., Hiemstra, D., Potthast, M., Zerhoudi, S.: The first international workshop on open web search (WOWS). In: Proc. of ECIR 2024, LNCS, vol. 14612, pp. 426–431, Springer (2024)
[26] Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: Splade v2: Sparse lexical and expansion model for information retrieval (2021), https://doi.org/10.48550/arXiv.2109.10086
[27] Formal, T., Piwowarski, B., Clinchant, S.: Splade: Sparse lexical and expansion model for first stage ranking. In: Proc. of SIGIR 2021, pp. 2288–2292 (2021)
[28] Fröbe, M., Mackenzie, J., Mitra, B., Nardini, F.M., Potthast, M.: ReNeuIR at SIGIR 2024: The third workshop on reaching efficiency in neural information retrieval. In: Proc. of SIGIR 2024, pp. 3051–3054, ACM (2024)
[29] Fröbe, M., Parry, A., Scells, H., Wang, S., Zhuang, S., Zuccon, G., Potthast, M., Hagen, M.: Corpus Subsampling: Estimating the Effectiveness of Neural Retrieval Models on Large Corpora. In: Proc. of ECIR 2025, pp. 453–471, LNCS, Springer (2025)
[30] Fröbe, M., Reimer, J., MacAvaney, S., Deckers, N., Reich, S., Bevendorff, J., Stein, B., Hagen, M., Potthast, M.: The Information Retrieval Experiment Platform. In: Proc. of SIGIR 2023, pp. 2826–2836, ACM (2023)
[31] Fröbe, M., Wiegmann, M., Kolyada, N., Grahm, B., Elstner, T., Loebe, F., Hagen, M., Stein, B., Potthast, M.: Continuous Integration for Reproducible Shared Tasks with TIRA.io. In: Proc. of ECIR 2023, pp. 236–241, LNCS, Springer (2023)

[32] Geng, Z., Wang, Y., Ru, D., Yang, Y.: Towards competitive search relevance for inference-free learned sparse retrievers (2025), https://doi.org/10.48550/arXiv.2411.04403
[33] Hagen, T., Fröbe, M., Merker, J.H., Scells, H., Hagen, M., Potthast, M.: Tirex tracker: The information retrieval experiment tracker. In: Proc. of SIGIR 2025, pp. 3764–3771, ACM (2025)
[34] Khandel, P., Yates, A., Varbanescu, A.L., de Rijke, M., Pimentel, A.D.: PEIR: modeling performance in neural information retrieval. In: Proc. of ECIR 2025, pp. 279–294, LNCS, Springer (2025)
[35] Lassance, C., Déjean, H., Formal, T., Clinchant, S.: Splade-v3: New baselines for splade (Mar 2024), https://doi.org/10.48550/arXiv.2403.06789
[36] Lawrie, D.J., Kayi, E.S., Yang, E., Mayfield, J., Oard, D.W.: PLAID SHIRTTT for large-scale streaming dense retrieval. In: Proc. of SIGIR 2024, pp. 2574–2578, ACM (2024)
[37] Lin, J., Ma, X.: A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques (Jun 2021), https://doi.org/10.48550/arXiv.2106.14807
[38] MacAvaney, S., Yates, A., Feldman, S., Downey, D., Cohan, A., Goharian, N.: Simplified data wrangling with ir_datasets. In: Proc. of SIGIR 2021, pp. 2429–2436, ACM (2021)
[39] Macdonald, C., Tonellotto, N., MacAvaney, S., Ounis, I.: PyTerrier: Declarative experimentation in Python from BM25 to dense retrieval. In: Proc. of CIKM 2021, pp. 4526–4533, ACM (2021)
[40] Mackenzie, J., Mallia, A., Moffat, A., Petri, M.: Accelerating learned sparse indexes via term impact decomposition. In: Proc. of EMNLP 2022, pp. 2830–2842 (2022)
[41] Mackenzie, J., Trotman, A., Lin, J.: Wacky weights in learned sparse representations and the revenge of score-at-a-time query evaluation (Oct 2021), https://doi.org/10.48550/arXiv.2110.11540
[42] Mallia, A., Mackenzie, J., Suel, T., Tonellotto, N.: Faster learned sparse retrieval with guided traversal. In: Proc. of SIGIR 2022, pp. 1901–1905 (2022)
[43] Mallia, A., Ottaviano, G., Porciani, E., Tonellotto, N., Venturini, R.: Faster blockmax WAND with variable-sized blocks. In: Proc. of SIGIR 2017, pp. 625–634, ACM (2017)
[44] Mallia, A., Siedlaczek, M., Mackenzie, J., Suel, T.: PISA: performant indexes and search for academia. In: Proc. of OSIRRC@SIGIR 2019, pp. 50–56 (2019), URL http://ceur-ws.org/Vol-2409/docker08.pdf
[45] Nardini, F.M., Nguyen, T., Rulli, C., Venturini, R., Yates, A.: Effective inference-free retrieval for learned sparse representations. In: Proc. of SIGIR 2025, pp. 2936–2940 (2025)
[46] Nguyen, T., MacAvaney, S., Yates, A.: A unified framework for learned sparse retrieval. In: Proc. of ECIR 2023, pp. 101–116 (Apr 2023)
[47] Qiao, Y., Yang, Y., He, S., Yang, T.: Representation sparsification with hybrid thresholding for fast splade-based document retrieval. In: Proc. of SIGIR 2023, pp. 2329–2333 (2023)

[48] Raasveldt, M., Mühleisen, H.: Duckdb: an embeddable analytical database. In: Proc. of SIGMOD 2019, pp. 1981–1984, ACM (2019)
[49] Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at trec-3. In: Proc. of TREC 1994, NIST (1994)
[50] Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter (Mar 2020), https://doi.org/10.48550/arXiv.1910.01108
[51] Scells, H., Zhuang, S., Zuccon, G.: Reduce, reuse, recycle: Green information retrieval research. In: Proc. of SIGIR 2022, pp. 2825–2837 (2022)
[52] Schlatt, F., Fröbe, M., Hagen, M.: Lightning IR: Straightforward Fine-tuning and Inference of Transformer-based Language Models for Information Retrieval. In: Proc. of WSDM 2025, pp. 1048–1051, ACM (2025)
[53] Shen, X., Geng, Z., Yang, Y.: Exploring ℓ_0 sparsification for inference-free sparse retrievers. In: Proc. of SIGIR 2025, pp. 2572–2576 (2025)
[54] Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. Journal of documentation **28**(1), 11–21 (1972)
[55] Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Proc. of ACL 2019, pp. 3645–3650, ACL (2019)
[56] Turtle, H., Flood, J.: Query evaluation: Strategies and optimizations. Information Processing & Management **31**(6), 831–850 (1995)
[57] Upadhyay, S., Pradeep, R., Thakur, N., Campos, D., Craswell, N., Soboroff, I., Dang, H.T., Lin, J.: A large-scale study of relevance assessments with large language models: An initial look. CoRR **abs/2411.08275** (2024), https://doi.org/10.48550/ARXIV.2411.08275
[58] Voorhees, E.M.: The philosophy of information retrieval evaluation. In: Proc. of CLEF 2001, LNCS, vol. 2406, pp. 355–370, Springer (2001)
[59] Voorhees, E.M.: Overview of the TREC 2004 Robust track. In: Proc. of TREC 2004, NIST (2004)
[60] Voorhees, E.M.: The evolution of cranfield. In: Proc. of CLEF 2019, pp. 45–69, Springer (2019)
[61] Xiao, S., Liu, Z., Zhang, P., Muennighoff, N., Lian, D., Nie, J.Y.: C-pack: Packaged resources to advance general chinese embedding (May 2024), https://doi.org/10.48550/arXiv.2309.07597
[62] Zamani, H., Dehghani, M., Croft, W.B., Learned-Miller, E., Kamps, J.: From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In: Proc. of CIKM 2018, pp. 497–506 (2018)