Operational Advice for Dense and Sparse Retrievers: HNSW, Flat, or Inverted Indexes?

Jimmy Lin

David R. Cheriton School of Computer Science
University of Waterloo

jimmylin@uwaterloo.ca

Abstract

Practitioners working on dense retrieval today face a bewildering number of choices. Beyond selecting the embedding model, another consequential choice is the actual implementation of nearest-neighbor vector search. While best practices recommend HNSW indexes, flat vector indexes with brute-force search represent another viable option, particularly for smaller corpora and for rapid prototyping. In this paper, we provide experimental results on the BEIR dataset using the open-source Lucene search library that explicate the tradeoffs between HNSW and flat indexes (including quantized variants) from the perspectives of indexing time, query evaluation performance, and retrieval quality. With additional comparisons between dense and sparse retrievers, our results provide guidance for today's search practitioner in understanding the design space of dense and sparse retrievers. To our knowledge, we are the first to provide operational advice supported by empirical experiments in this regard.

1 Introduction

Retrieval-augmented generation (RAG), which involves injecting search results into the prompt of a large language model (LLM) to provide context or "grounding", is one of the most popular and effective generative AI techniques today (Lewis et al., 2020; Gao et al., 2024). It is widely recognized that the quality of the generated responses depends to a large extent on the quality of the search results, i.e., "garbage in, garbage out". This makes retrieval a critical component of RAG.

Today, practitioners typically take advantage of vector search to generate search results, but they face a bewildering number of choices. There's first-stage retrieval to generate a list of candidates, possibly followed by reranking. Even focused on the first stage, dense retrieval models and sparse retrieval models compete for attention, often confusing newcomers; and this is only considering single-vector variants, leaving aside multi-vector techniques such as ColBERT (Khattab and Zaharia, 2020). To offer a conceptual structure, Lin (2021) provides a framework for thinking about retrieval in terms of nearest-neighbor search over vector representations (of queries and documents), where these representations can be dense (typically called embeddings, generated from transformers) or sparse (also generated by transformers). Relevance is captured by simple vector operations such as the dot product, and a retriever's task is to efficiently produce the top-k documents from a corpus based on these similarity comparisons.

The focus of most efforts today lie in dense retrieval models (Karpukhin et al., 2020), where queries and documents are represented by dense vectors (i.e., embeddings), typically generated by transformer models that have been fine-tuned on human-labeled or synthetically generated relevance data. This forms the starting point of our work. Nearest-neighbor search over dense representation vectors defines rankings of documents with respect to queries, but says nothing about how those rankings are computed efficiently at scale. Presently, best practices recommend the use of hierarchical navigable small-world network (HNSW) indexes (Malkov and Yashunin, 2020). An alternative is so-called flat indexes that take advantage of brute-force search, which are attractive in certain scenarios. But when? More broadly, a search practitioner today faces choices between dense retrieval models and sparse retrieval models. How do they navigate these options?

This work attempts to sort through these myriad options for dense and sparse retrievers, in particular focusing on three research questions:

RQ1 For dense retrieval, when should HNSW indexes be used vs. flat indexes and what are the associated tradeoffs?

RQ2 For both HNSW and flat indexes, when

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 865–872

should quantization be applied and what are the associated tradeoffs?

RQ3 More broadly, what are the effectiveness–efficiency tradeoffs between dense and sparse retrieval across different corpora?

Ultimately, our goal is to provide operational guidance for a search practitioner to navigate the complex design space of dense and sparse retrieval. Our goal is to explicate the tradeoffs involving indexing time, query evaluation performance, and retrieval quality to help practitioners make better decisions informed by experimental evidence.

2 The State of the Art

It makes sense to begin with a characterization of the state of the art, not in the sense of leaderboard chasing, but the day-to-day choices faced by search practitioners "in the real world". Naturally, it is not possible to cover *all* aspects of retrieval, so we focus on the three main research questions articulated in the introduction.

Brute-force search with flat indexes was introduced in Elasticsearch v8.13 (released March 2024). As Elasticsearch is built on the Lucene search library used in our experiments, it provides an appropriate starting point for our discussions. An official blog post[1] accompanying the release offers the following advice: "when the size of the set... is rather small, it is usually better to rely on brute-force vector search rather than on HNSW-based vector search." But what does "rather small" mean? And what other factors matter? This advice cannot be easily operationalized, making it unhelpful for search practitioners (RQ1). Elsewhere, we find advocates for flat indexes using DataFrames, or even Numpy,[2] especially for rapid prototyping. The same Elasticsearch blog post discusses int8 quantization, but is similarly vague about specific guidance (RQ2). Finally, "heads up" fair comparisons between dense retrievers and alternative models are difficult to find (RQ3).

While we point to this specific instance to illustrate a gap in the state of the art, the general sentiments expressed in the blog post are not unique. Other documents found on the web and on social media are similarly handy-wavy in providing guidance, and what few specifics offered are unsupported by empirical evidence. To our knowledge, the concrete advice offered in this paper using an existing, widely adopted benchmark does not exist anywhere else, and represents the contribution of our work. Of course, specific application deployments require balancing many competing factors, and it is impossible to offer "one-size-fits-all" advice. Nevertheless, we provide empirical evidence that accurately characterizes the design space to inform system builders.

It is obvious that performance is affected by scale (e.g., size of corpora and length of individual documents), the embedding model, the types of queries, as well as many other factors, but it would be desirable to provide search practitioners today more specific guidance. According to a talk by Chroma, a vector database vendor,[3] most of their customers manage corpora ranging from "several hundreds of thousands" to "several millions" vectors. This is consistent with other discussions on social media, and provides us a point of calibration. We structure our study in terms of corpora in this range of sizes to benefit the broadest audience.

3 Methods

We begin by describing and justifying our experimental setup. All experiments in this paper take advantage of BEIR (Thakur et al., 2021), which comprises a large collection of individual retrieval datasets and has emerged as the standard benchmark for evaluating retrieval applications. We provide detailed experimental results over 29 different individual datasets,[4] each with different corpora, queries, and task definitions. This variety provides a cross section of search tasks and realistically reflects real-world scenarios.

Our evaluations were conducted with the open-source Lucene search library, a choice that deserves some discussion and justification. We provide two main reasons: First, Lucene is the most widely deployed search library in the world, mostly via platforms such as Elasticsearch, Solr, and OpenSearch. Devins et al. (2022) have shown that implementations in Lucene simplify many aspects of IR experiments, but yet can be easily ported over to Elasticsearch—this combination facilitates prototyping while preserving fidelity to real-world sce-

[1] https://www.elastic.co/blog/whats-new-elasticsearch-platform-8-13-0

[2] https://x.com/softwaredoug/status/1802433164201415000

[3] https://www.youtube.com/watch?v=E4ot5d79jdA

[4] Note that CQADupStack is actually comprised of 12 different "verticals".

narios. Thus, our results would be of broad interest to many practitioners in the community.

Second, our work with Lucene provides a comparison across dense and sparse techniques that is as fair as possible given currently available software. While Lucene provides a production-grade implementation of HNSW indexes, it is one of many existing options currently available on the market. Faiss (Johnson et al., 2019) is another popular option, and there is a vibrant ecosystem of vendors providing vector search capabilities (Weaviate, Chroma, Pinecone, Vectara, Vespa, and many others). Vector search has also been integrated into relational databases (Xian et al., 2024), for example, pgvector for Postgres.

However, we selected Lucene because it provides implementations of *both* dense and sparse retrieval, making comparisons reasonably fair. For example, comparing Faiss HNSW indexes (implemented in C++) with Lucene inverted indexes (implemented in Java) or even Numpy would be conflating too many non-relevant factors (e.g., language choice). Within the same project (Lucene), we would expect different retrieval techniques to have comparable implementation quality. While Vespa does provide dense and sparse vector search capabilities, it remains niche and lacks the wide install base of Lucene, making results of limited interest to the broader community.

Retrieval models. We examined the following retrieval models in this study: (1) BGE `bge-base-en-v1.5` (Xiao et al., 2024) was selected as a representative dense retrieval model. (2) SPLADE++ EnsembleDistil (ED) (Formal et al., 2022) was selected as a representative sparse retrieval model. (3) BM25 (Robertson and Zaragoza, 2009) provides the baseline; here we use the variant where all document fields are concatenated prior to indexing (Kamalloo et al., 2024).

For the dense retrieval model (BGE), our work examined two index types. First, we considered hierarchical navigable small-world network (HNSW) indexes (Malkov and Yashunin, 2020), which represent best practices today for nearest-neighbor search over dense vectors. Most "vector DB" vendors today offer variants of such indexes.

Alternatively, we evaluated so-called "flat" indexes, where the dense vectors are simply stored sequentially, one after the other. "Indexing" in this case is simply rewriting the embedding vectors in an internal representation. Top-k retrieval is implemented as brute-force search: the retriever simply scans the vectors, computing (in our case) the dot product between the query and each document vector, retaining only the top k results.

For SPLADE++ ED, we used standard inverted indexes, taking advantage of the widely known "fake words" trick, where quantized impact scores replace the term frequency component in the postings, and query evaluation uses a "sum of term frequencies" scoring function. See Mackenzie et al. (2022) for more details. BM25 also used standard inverted indexes.

Implementation details. All experiments were conducted using the Anserini open-source IR toolkit (Yang et al., 2018), based on Lucene 9.9.1 (released Dec. 2023). We used bindings for Lucene HNSW indexes recently introduced in Ma et al. (2023). We set the HNSW indexing parameters `M` to 16 and `efC` to 100, both representing typical configurations. Lucene's HNSW indexing implementation generates different index segments and then merges them as needed in a hierarchical manner; we used all default settings here. On the retrieval end, we set `efSearch` to 1000, another common setting. The flat index implementation in Anserini is adapted from Elasticsearch.

All experiments were performed on a circa-2022 Mac Studio with an M1 Ultra processor containing 20 cores (16 performance, 4 efficiency) and 128 GB memory, running macOS Sonoma 14.5 and OpenJDK 21.0.2. We enabled the `jdk.incubator.vector` module for more efficient vector operations. Both indexing and retrieval experiments used 16 threads. In all cases (HNSW, flat, and inverted indexes), we retrieved 1000 hits and evaluated retrieval quality in terms of nDCG@10, per BEIR guidelines. Query evaluation performance was measured in terms of queries per second (QPS) using 16 threads.

4 Experimental Results

We begin with a comparison between flat, HNSW, and inverted indexes in terms of effectiveness and efficiency, shown in Table 1. Each row captures a dataset from BEIR. The rows are sorted by the size of each corpus (number of documents, $|\mathcal{C}|$), so scanning down the rows, we encounter datasets of increasing size. The table is informally divided into three sections that we characterize as "small" (less then 100K documents), "medium" (between 100K and 1M), and "large" (more than 1M). The column marked $|Q|$ shows the number of queries in each

Dataset	$\|\mathcal{C}\|$	$\|Q\|$	nDCG@10			QPS (cached)			QPS (ONNX)			QPS
			Dense	Sparse	BM25	Flat	HNSW	INV	Flat	HNSW	INV	BM25
NFCorpus	3,633	323	0.373	0.347	0.322	270	280	430	210	200	220	480
SciFact	5,183	300	0.741	0.704	0.679	260	260	280	200	190	140	280
ArguAna	8,674	1,406	0.636	0.520	0.397	440	430	320	240	260	23	360
CQA Mathematica	16,705	804	0.316	0.238	0.202	330	340	350	240	240	210	390
CQA webmasters	17,405	506	0.406	0.317	0.306	320	330	290	210	220	180	340
CQA Android	22,998	699	0.507	0.390	0.380	310	320	350	220	220	190	380
SCIDOCS	25,657	1,000	0.217	0.159	0.149	290	330	330	240	230	190	190
CQA programmers	32,176	876	0.424	0.340	0.280	340	390	350	220	230	200	390
CQA GIS	37,637	885	0.413	0.315	0.290	350	360	380	220	230	190	380
CQA physics	38,316	1,039	0.472	0.360	0.321	360	360	410	220	230	200	420
CQA English	40,221	1,570	0.486	0.408	0.345	400	430	440	230	240	200	480
CQA stats	42,269	652	0.373	0.299	0.271	290	310	350	200	210	180	340
CQA gaming	45,301	1,595	0.597	0.496	0.482	410	430	430	230	240	210	460
CQA UNIX	47,382	1,072	0.422	0.317	0.275	360	360	410	210	230	200	390
CQA Wordpress	48,605	541	0.355	0.273	0.248	310	350	310	190	200	180	320
FiQA-2018	57,638	648	0.406	0.347	0.236	290	330	300	190	220	170	340
CQA tex	68,184	2,906	0.311	0.253	0.224	400	480	520	210	240	220	490
TREC-COVID	171,332	50	0.781	0.727	0.595	66	100	65	58	76	52	92
Touché 2020	382,545	49	0.257	0.247	0.442	38	85	52	33	61	47	68
Quora	522,931	10,000	0.889	0.834	0.789	75	200	420	61	110	180	770
Robust04	528,155	249	0.447	0.468	0.407	57	150	150	48	89	86	110
TREC-NEWS	594,977	57	0.443	0.415	0.395	29	72	54	27	67	47	47
NQ	2,681,468	3,452	0.541	0.538	0.305	15	140	130	14	90	85	470
Signal-1M	2,866,316	97	0.289	0.301	0.330	8.8	60	59	8.5	41	46	180
DBpedia	4,635,922	400	0.407	0.437	0.318	7.7	72	80	7.4	52	63	300
HotpotQA	5,233,329	7,405	0.726	0.687	0.633	7.6	74	69	7.4	52	46	460
FEVER	5,416,568	6,666	0.863	0.788	0.651	7.3	63	65	7.2	47	49	470
Climate-FEVER	5,416,593	1,535	0.312	0.230	0.165	7.1	62	73	6.9	44	47	290
BioASQ	14,914,603	500	0.415	0.498	0.522	2.6	56	24	2.6	40	23	210

Table 1: Main results comparing flat and HNSW indexes (BGE) and inverted indexes (SPLADE and BM25) in terms of effectiveness (nDCG@10) and query evaluation performance (queries per second, QPS). For nDCG@10, "Dense" refers to BGE and "Sparse" refers to SPLADE; "INV" refers to inverted indexes.

dataset; note that performance measurements are noisier with fewer queries. The next three columns show the effectiveness of the dense model (BGE), the sparse model (SPLADE), and BM25.

Query evaluation performance is captured in terms of queries per second (QPS). Due to the inherent noise in these measurements, we only report figures to two significant digits because any addition precision is unlikely to be meaningful. Our experiments are divided into two conditions, cached queries and "on-the-fly" query encoding using ONNX (not applicable to BM25). With cached queries, we are *not* measuring the latency associated with query encoding, whereas with ONNX, latency includes query encoding. These two measurements bookend the performance range: our ONNX encoding is performed on the CPU, and hence can be accelerated with GPU inference, but performance cannot exceed the cached condition. More details about ONNX integration in Anserini are discussed in Chen et al. (2023).

Obviously, in production settings, query evaluation performance must necessarily include query encoding, as the system does not know the queries in advance. However, in a prototyping setting, or when running benchmarks repeatedly, it makes sense to cache the query representations. Thus, we believe that both ways of measuring performance are informative, but for different scenarios.

4.1 Flat vs. HNSW Indexes

RQ1 For dense retrieval, when should HNSW indexes be used vs. flat indexes and what are the associated tradeoffs?

Table 1 provides guidance for this research question, illustrated with the BGE model. Most pertinent is the comparison between flat and HNSW indexes under the "cached" and "ONNX" conditions. We make the following observations:

- For "small" corpora less than 100K documents, there appear to be negligible differences between flat and HNSW indexes. For example, in an exploratory or prototyping setting, we do not see the differences in QPS as meaningful.

- For "medium" corpora (between 100K and 1M), the performance differences between flat indexes and HNSW indexes become larger: very roughly, flat indexes are 2–3× slower with cached query

Dataset	$\|\mathcal{C}\|$	Index Time Flat	Index Time HNSW	nDCG@10	nDCG@10 avg Δ	nDCG@10 min, max
TREC-COVID	171k	0.9	1.8	0.781	0.000	[0.000, 0.000]
Touché 2020	383k	1.0	1.9	0.257	0.000	[0.000, 0.000]
Quora	523k	1.0	2.4	0.889	0.000	[0.000, 0.000]
Robust04	528k	1.0	2.1	0.447	0.001	[0.000, 0.001]
TREC-NEWS	595k	1.0	2.1	0.443	0.001	[−0.004, 0.007]
NQ	2.7m	2.4	15.6	0.541	0.002	[0.001, 0.003]
Signal-1M	2.9m	2.3	14.5	0.289	0.010	[0.006, 0.013]
DBpedia	4.6m	3.2	31.5	0.407	0.001	[−0.001, 0.004]
HotpotQA	5.2m	4.1	33.3	0.726	0.010	[0.009, 0.011]
FEVER	5.4m	4.2	35.0	0.863	0.005	[0.004, 0.006]
Climate-FEVER	5.4m	4.1	35.2	0.312	0.000	[0.000, 0.000]
BioASQ	14.9m	10.1	76.3	0.415	0.015	[0.011, 0.020]

Table 2: Comparing flat vs. HNSW indexes using BGE. Indexing times are reported in minutes. The "avg Δ" column reports the average degradation of HNSW scores over five trials; min/max report the observed min and max values across the trials; negative values indicate that HNSW indexes achieved higher scores.

representations, but after factoring in query encoding (ONNX), the gap is reduced. For a practitioner prototyping with a small set of queries, we would recommend flat indexes, since operationally, the QPS differences are likely not meaningful. As an example, on TREC-NEWS, the wallclock difference in evaluating on the set of 57 queries is around a second at the most.

- For "large" corpora (more than 1M), the performance differences can be quite large: flat indexes are up to an order of magnitude slower than HNSW indexes for corpora in the 2M–5M documents range, and even slower for BioASQ, the largest BEIR corpora, at ∼15M documents.

To more fully characterize these tradeoffs, we need to examine two additional aspects of the design space: indexing time and retrieval quality. Once again, we focus on the BGE dense retrieval model. In Table 2, the columns "Flat" and "HNSW" compare indexing time, averaged over five trials, rounded to the nearest tenth of a minute. Rows are sorted by increasing size, same as in Table 1. For brevity, we omit results for small corpora, where the indexing times are for the most part well under a minute and the results are uninteresting.

For medium corpora (under 1M documents), we argue that the differences in indexing times are not meaningful, but the differences appear to grow as the corpus size increases; for corpora with more than 1M documents, the HNSW indexing time can be several times longer. With flat indexes, "indexing" simply involves reading input vectors and rewriting them in Lucene's internal representation. On the other hand, Lucene's HNSW indexing implementation requires building traversal graphs over segments of documents and then hierarchically merging them; indexing time does not appear to be linear with respect to corpus size.

The retrieval quality (effectiveness) implications of flat vs. HNSW indexes using the BGE embedding model are also shown in Table 2, in the columns grouped under nDCG@10. The scores are the same as in Table 1, under the "Dense" column. Flat indexes, which yield exact similarity scores, provide the ground truth reference. Since HNSW indexes enable fast *approximate* nearest-neighbor search, there is typically some effectiveness degradation, i.e., scores from HNSW indexes are usually lower. Furthermore, since HNSW index construction is non-deterministic, scores from each trial may differ slightly. The "avg Δ" column reports the average degradation of HNSW scores over five trials. The "min" and "max" columns report the observed min and max values across the trials; negative values indicate that a particular HNSW trial achieved a higher score than the corresponding flat index (sometimes possible).

Tables 1 and 2 together characterize the tradeoffs between flat and HNSW indexes. For "medium" corpora, HNSW indexing is slower than flat indexing, but we argue that the differences are not meaningful. There are also some effectiveness differences, but they are mostly small. For "large" corpora (more than 1M documents), we see interesting tradeoffs in indexing time versus query evaluation performance. The much higher QPS we report in Table 1 comes at a large cost in indexing time; HNSW indexes can take much longer to build than flat indexes. Also, we observe that retrieval quality degrades more as corpus size increases.

4.2 The Impact of Quantization

RQ2 For both HNSW and flat indexes, when should quantization be applied and what are the associated tradeoffs?

Here, we examine flat and HNSW indexes separately. Results comparing flat and quantized (int8) flat indexes are reported in Table 3. Our analysis is organized into three relevant factors, as before: indexing time, query evaluation performance (QPS), and retrieval quality (nDCG@10). Note that index quantization in Lucene is *not* deterministic, and we report figures averaged across five trials. The reference indexing times for flat indexes are copied from Table 2 (measured in minutes), with

Dataset	$\lvert\mathcal{C}\rvert$	Index Time	Δ	QPS (Cached)	Δ	QPS (ONNX)	Δ	nDCG@10	avg Δ	min	max
TREC-COVID	171,332	0.9	∼	66	+3.8%	58	∼	0.781	−0.003	[−0.003	−0.002]
Touché 2020	382,545	1.0	∼	38	+31%	33	+25%	0.257	0.007	[0.006	0.008]
Quora	522,931	1.0	+6%	75	+26%	61	+15%	0.889	0.001	[0.001	0.001]
Robust04	528,155	1.0	∼	57	+28%	48	+21%	0.447	0.001	[−0.001	0.001]
TREC-NEWS	594,977	1.0	∼	29	+48%	27	+48%	0.443	0.009	[0.007	0.012]
NQ	2,681,468	2.4	∼	15	+35%	14	+29%	0.541	0.002	[0.002	0.003]
Signal-1M	2,866,316	2.3	+10%	8.8	+63%	8.5	+62%	0.289	0.004	[0.002	0.006]
DBpedia	4,635,922	3.2	+17%	7.7	+47%	7.4	+45%	0.407	−0.001	[−0.002	0.000]
HotpotQA	5,233,329	4.1	+14%	7.6	+36%	7.4	+33%	0.726	0.000	[0.000	0.000]
FEVER	5,416,568	4.2	+13%	7.3	+36%	7.2	+33%	0.863	0.001	[0.000	0.001]
Climate-FEVER	5,416,593	4.1	+15%	7.1	+39%	6.9	+38%	0.312	0.003	[0.002	0.004]
BioASQ	14,914,603	10.1	+12%	2.6	+38%	2.6	+37%	0.415	0.003	[0.003	0.003]

Table 3: The effects of (int8) quantization for flat indexes, compared to non-quantized versions.

Dataset	$\lvert\mathcal{C}\rvert$	Index Time	Δ	QPS (Cached)	Δ	QPS (ONNX)	Δ	nDCG@10	avg Δ	min	max
TREC-COVID	171,332	1.8	∼	100	∼	76	∼	0.781	−0.003	[−0.003	−0.002]
Touché 2020	382,545	1.9	∼	85	+6%	61	+11%	0.257	0.006	[0.006	0.007]
Quora	522,931	2.4	∼	200	+44%	110	+29%	0.889	0.001	[0.001	0.001]
Robust04	528,155	2.1	∼	150	+21%	89	+22%	0.447	0.001	[−0.001	0.003]
TREC-NEWS	594,977	2.1	∼	72	+22%	67	∼	0.443	0.011	[0.009	0.013]
NQ	2,681,468	15.6	+33%	140	+47%	90	+29%	0.541	0.003	[0.002	0.004]
Signal-1M	2,866,316	14.5	+46%	60	+57%	41	+63%	0.289	0.010	[0.007	0.015]
DBpedia	4,635,922	31.5	+55%	72	+76%	52	+58%	0.407	−0.001	[−0.004	0.000]
HotpotQA	5,233,329	33.3	+60%	74	+130%	52	+90%	0.726	0.018	[0.016	0.019]
FEVER	5,416,568	35.0	+73%	63	+143%	47	+104%	0.863	0.010	[0.008	0.012]
Climate-FEVER	5,416,593	35.2	+79%	62	+142%	44	+98%	0.312	0.001	[0.000	0.002]
BioASQ	14,914,603	76.3	+5%	56	+29%	40	+25%	0.415	0.017	[0.011	0.024]

Table 4: The effects of (int8) quantization for HNSW indexes, compared to non-quantized versions. Note that exact rankings from flat indexes provide the reference nDCG@10 scores.

Δ reporting the increase in indexing time due to quantization (as a percentage). Similarly, query performance (QPS) under the cached and ONNX conditions are copied from Table 1 for the reference (non-quantized) condition: the Δ columns show increases in QPS from quantization. In the table, ∼ refers to differences less than 5%, since our measurements are noisy and we do not wish to draw attention to small differences that are likely not meaningful. Overall, we see that quantization provides a big boost in performance (QPS) at a relatively low cost in additional indexing time.

Finally, nDCG@10 differences are organized in the same way as in Table 2, where we report average, min, and max with respect to (non-quantized) flat indexes. Negative values indicate that quantization *increased* effectiveness (possible in some cases). Nevertheless, quantization has a relatively minor impact on retrieval quality overall.

Results comparing HNSW and quantized (int8) HNSW indexes are reported in Table 4, which is organized in the same manner as Table 3. Note, however, that the reference nDCG@10 scores here are taken from exact rankings using flat indexes. This means that the measure of degradation includes *both* HNSW indexing and quantization.

For HNSW indexes, we observe quantization tradeoffs that are different from flat indexes. With medium corpora, there does not appear to be meaningful increases in indexing time, but with large corpora, indexing appears to be much slower. Interestingly, for BioASQ, the increase in indexing time is only marginal,[5] which suggests that the additional costs associated with quantization are masked by other components of the indexing pipeline.

Quantization for HNSW indexes, however, delivers large benefits in increased QPS, even more than for quantization in flat indexes. The effectiveness degradation of quantized HNSW indexes is comparable to non-quantized HNSW indexes, which suggests that the effectiveness impact of quantization is minor at most.

4.3 Dense Retrieval in a Broader Context

RQ3 More broadly, what are the effectiveness–efficiency tradeoffs between dense and sparse retrieval across different corpora?

Effectiveness comparisons of dense and sparse retrieval abound in the literature. Overall, one approach does not appear to be dominant, and it might be fair to characterize dense and sparse models as comparable in terms of effectiveness.

[5] Nope, verified that this isn't a bug.

However, query evaluation performance has received little attention by researchers, and we contribute a comparison between SPLADE++ ED and BGE in a fair setting, shown in Table 1. In terms of QPS, both appear to be comparable, looking at the "HNSW" vs. "INV" columns.[6] There does not appear to be a compelling reason to choose dense retrieval over sparse retrieval (or vice versa) from the performance point of view. Indeed, the literature is consistent in advocating hybrid techniques that combine both approaches (Thakur et al., 2021; Ma et al., 2022; Kamalloo et al., 2024).

Table 1 also provides a comparison between SPLADE++ ED and BM25. In terms of effectiveness, the SPLADE model dominates BM25 and outperforms it for nearly all of the datasets in BEIR; the exceptions are Touché, Signal-1M, and BioASQ. In the first case, Thakur et al. (2024) provides a detailed error analysis explaining these results. However, we see from the final column that BM25 is much faster than SPLADE++ ED; the difference is close to an order of magnitude in the case of BioASQ, the largest corpus. For some points in the effectiveness–efficiency tradeoff space, there is still a role for BM25.

5 Discussion

The primary goal of this paper is to replace "hand waving" with empirical evidence for the benefit of search practitioners. Our experimental results illustrate the tradeoff space with BEIR, a widely adopted retrieval benchmark. While the ultimate choices of system builders will depend on the real-world scenario (from prototyping to proof of concept to production deployment), we can offer some advice. At a high level, for corpora with fewer than 1M documents, we do not see a compelling advantage to using HNSW indexes. For larger corpora, however, we feel that the advantages of HNSW indexes in terms of query evaluation performance offset the downsides.

Another issue worth discussing is the retrieval quality degradation associated with HNSW indexing and quantization. These factors are not typically discussed in academic research, but are important from the perspective of building real-world systems. A recap of the issues: both HNSW indexing and quantization are non-deterministic and typically degrade retrieval effectiveness with respect to exact similarity comparisons (captured in flat indexes). As an example, BioASQ results from Table 4 show that, with HNSW and quantization, nDCG@10 scores are 0.017 lower (averaged across five trials), with a max difference of 0.024; this translates into 4.1% and 5.8%, respectively—relatively large differences. These effects are potentially problematic when comparing different embedding models that are "close" in terms of quality, because it would be hard to tease apart model quality from an "unlucky" sub-optimal index. Nearly all academic papers sweep these differences under the rug, but they represent important practical considerations. For this reason, flat indexes are appealing for rapid prototyping in order to isolate the quality of embedding models.

[6] ArguAna appears to be an outlier for SPLADE; we verified that this was not a bug.

6 Conclusions

There are three main limitations to this work worth pointing out. First, we study only a single instance of a dense and sparse retrieval model (BGE and SPLADE++ ED). While both are popular and representative, there are many other models worth considering and new ones appearing all the time. Second, we only evaluate performance on a single system. An exhaustive matrix experiment involving different models and systems (architectures, OSes, etc.) would be impractical, and we expect the broad contours of our findings to remain invariant. However, more experiments are needed to confirm the generalizability of our findings.

Another limitation is our focus on Lucene, even though there are many other HNSW implementations available. This issue has already been discussed in Section 3, and it may be the case that other system combinations will alter our conclusions. However, as we pointed out, such comparisons are difficult to set up in a fair manner. Nevertheless, the dominance of Lucene means that our findings are of broad interest, worthy of consideration even for users of other platforms.

There are many more decisions that a search practitioner needs to make when building a full RAG system, beyond the explicit research questions that we consider in this work. For example, what are the roles of reranking and prompt engineering? How do we deal with dynamically changing documents? The list goes on. Nevertheless, we hope that this work offers a starting point in providing empirically grounded guidance for search practitioners building real-world applications.

Acknowledgements

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC). We'd like to acknowledge Snowflake for additional funding. Thanks to Steven Chen for helpful comments on an earlier draft of this paper.

References

Haonan Chen, Carlos Lassance, and Jimmy Lin. 2023. End-to-end retrieval with learned dense and sparse representations using Lucene. *arXiv:2311.18503*.

Josh Devins, Julie Tibshirani, and Jimmy Lin. 2022. Aligning the research and practice of building search applications: Elasticsearch and Pyserini. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining (WSDM 2022)*.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From distillation to hard negative sampling: Making sparse neural IR models more effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *arXiv:2312.10997*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Ehsan Kamalloo, Nandan Thakur, Carlos Lassance, Xueguang Ma, Jheng-Hong Yang, and Jimmy Lin. 2024. Resources for brewing BEIR: Reproducible reference models and statistical analyses. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33*, pages 9459–9474.

Jimmy Lin. 2021. A proposed conceptual framework for a representational approach to information retrieval. *arXiv:2110.01529*.

Xueguang Ma, Kai Sun, Ronak Pradeep, Minghan Li, and Jimmy Lin. 2022. Another look at DPR: Reproduction of training and replication of retrieval. In *Proceedings of the 44th European Conference on Information Retrieval (ECIR 2022), Part I*.

Xueguang Ma, Tommaso Teofili, and Jimmy Lin. 2023. Anserini gets dense retrieval: Integration of Lucene's HNSW indexes. In *Proceedings of the 32nd International Conference on Information and Knowledge Management (CIKM 2023)*.

Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2022. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Transactions on Information Systems*, 41:Article No. 96.

Yu A. Malkov and D. A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Nandan Thakur, Luiz Bonifacio, Maik Fröbe, Alexander Bondarenko, Ehsan Kamalloo, Martin Potthast, Matthias Hagen, and Jimmy Lin. 2024. Systematic evaluation of neural retrieval models on the Touché 2020 argument retrieval subset of BEIR. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*.

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 *Proceedings of NeurIPS 2021, Datasets and Benchmarks*.

Jasper Xian, Tommaso Teofili, Ronak Pradeep, and Jimmy Lin. 2024. Vector search with OpenAI embeddings: Lucene is all you need. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM 2024)*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packaged resources to advance general Chinese embedding. *arXiv:2309.07597*.

Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality*, 10(4):Article 16.