Vector Search with OpenAI Embeddings: Lucene Is All You Need

Jasper Xian
University of Waterloo
Ontario, Canada

Tommaso Teofili
Roma Tre University
Italy

Ronak Pradeep
University of Waterloo
Ontario, Canada

Jimmy Lin
University of Waterloo
Ontario, Canada

ABSTRACT

We provide a reproducible, end-to-end demonstration of vector search with OpenAI embeddings using Lucene on the popular MS MARCO passage ranking test collection. The main goal of our work is to challenge the prevailing narrative that a dedicated vector store is necessary to take advantage of recent advances in deep neural networks as applied to search. Quite the contrary, we show that hierarchical navigable small-world network (HNSW) indexes in Lucene are adequate to provide vector search capabilities in a standard bi-encoder architecture. This suggests that, from a simple cost–benefit analysis, there does not appear to be a compelling reason to introduce a dedicated vector store into a modern "AI stack" for search, since such applications have already received substantial investments in existing, widely deployed infrastructure.

CCS CONCEPTS

• **Information systems** → **Search engine architectures and scalability**.

KEYWORDS

dense vector search; HNSW indexes

ACM Reference Format:
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 '24), March 4–8, 2024, Merida, Mexico.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3616855.3635691

1 INTRODUCTION

Recent advances in the application of deep neural networks to search have focused on representation learning in the context of the so-called bi-encoder architecture, where content (queries, passages, and even images and other multimedia content) is represented by dense vectors (so-called "embeddings"). Dense retrieval models using this architecture form the foundation of retrieval augmentation in large language models (LLMs), a popular and productive approach to improving LLM capabilities in the broader context of generative AI [1, 23].

The dominant narrative today is that since dense retrieval requires the management of a potentially large number of dense vectors, enterprises require a dedicated "vector store" or "vector database" as part of their "AI stack". There exists a cottage industry of startups pitching vector stores as novel, must-have components in a modern enterprise architecture; examples include Pinecone, Weaviate, Chroma, Milvus, Qdrant, just to name a few.

We articulate a counterpoint to this narrative (cf. [3]). Our arguments center around a simple cost–benefit analysis: since search is a brownfield application, many organizations have already made substantial investments in these capabilities. Today, production infrastructure is dominated by the broad ecosystem centered around the open-source Lucene search library, most notably driven by platforms such as Elasticsearch, OpenSearch, and Solr. While the Lucene ecosystem has admittedly been slow to adapt to recent trends in representation learning, there are strong signals that serious investments are being made in this space. Thus, we see no compelling reason why separate, dedicated vector stores are necessary in a modern enterprise. In short, the benefits do not appear to justify the cost of additional architectural complexity.

It is important to separate *capabilities* from distinct *software components*. While hierarchical navigable small-world network (HNSW) indexes [22] represent the state of the art today in approximate nearest neighbor search—the most important operation for vector search using embeddings—it is not clear that providing operations around HNSW indexes requires a separate and distinct vector store. Indeed, the most recent major release of Lucene (version 9, from Dec 2021) includes HNSW indexing and vector search, and these capabilities have steadily improved over time. The open-source nature of the Lucene ecosystem means that advances in the core library itself will be rapidly adopted and integrated into other software platforms within the broader ecosystem.

The growing popularity of embedding APIs [11] further strengthens our arguments. These APIs encapsulate perhaps the most complex and resource-intensive aspect of vector search—the generation of dense vectors themselves. Embedding APIs hide model training, deployment, and inference behind the well-known benefits of service-based computing, much to the delight of practitioners. To support our arguments, we demonstrate vector search with OpenAI embeddings [24] using the popular MS MARCO passage test collection [2]: we have encoded the entire corpus and indexed the embedding vectors using Lucene. Evaluation on the MS MARCO development set queries and queries from the TREC DL Tracks [4, 5] show that OpenAI embeddings are able to achieve a respectable

WSDM '24, March 4–8, 2024, Merida, Mexico

ACM ISBN 979-8-4007-0371-3/24/03…$15.00
https://doi.org/10.1145/3616855.3635691

level of effectiveness. And as Devins et al. [6] have shown, anything doable in Lucene is relatively straightforward to replicate in Elasticsearch (and any other platform built on Lucene). Thus, we expect the ideas behind our demonstration to become pervasive in the near future. To facilitate broader adoption, we make available everything needed to reproduce our experiments, including the actual OpenAI embeddings.

2 ARCHITECTURE TO IMPLEMENTATION

The central idea behind the bi-encoder architecture is to encode queries and passages into dense vectors—commonly referred to as "embeddings"—such that relevant query–passage pairs receive high scores, computed as the dot product of their embeddings. In this manner, search can be formulated as a nearest neighbor search problem in vector space: given the query embedding, the system's task is to efficiently retrieve the top-k passage embeddings with the largest dot products [13]. Typically, "encoders" for generating the vector representations are implemented using transformers, which are usually fine-tuned in a supervised manner using a large dataset of relevant query–passage pairs [12, 28].

This formulation of search, in terms of comparisons between dense vectors, differs from "traditional" bag-of-words sparse representations that rely on inverted indexes for low-latency query evaluation. Instead, nearest neighbor search in vector space requires entirely different techniques: indexes based on hierarchical navigable small-world networks (HNSW) [22] are commonly acknowledged as representing the state of the art. The Faiss library [10] provides a popular implementation of HNSW indexes that is broadly adopted today and serves as a standard baseline. Despite conceptual similarities [13], it is clear that top-k retrieval on sparse vectors and dense vectors require quite different and distinct "software stacks". Since hybrid approaches that combine both dense and sparse representations have been shown to be more effective than either alone [18, 19], many modern systems combine separate retrieval components to achieve hybrid retrieval. For example, the Pyserini IR toolkit [15] integrates Lucene and Faiss for sparse and dense retrieval, respectively. Recognizing the need for managing both sparse and dense retrieval models, the dominant narrative today is that the modern enterprise "AI stack" requires a dedicated vector store or vector database, alongside existing fixtures such as relational databases, NoSQL stores, event stores, etc. This is the narrative that our work challenges.

Modern enterprise architectures are already exceedingly complex, and the addition of another software component (i.e., a distinct vector store) requires carefully weighing costs as well as benefits. The cost is obvious: increased complexity, not only from the introduction of a new component, but also from interactions with existing components. What about the benefits? While vector stores no doubt introduce new capabilities, the critical question is whether these capabilities can be provided via alternative means.

Search is a brownfield application. Wikipedia defines this as "a term commonly used in the information technology industry to describe problem spaces needing the development and deployment of new software systems in the immediate presence of existing (legacy) software applications/systems." Additionally, "this implies that any new software architecture must take into account and coexist with live software already in situ." Specifically, many organizations have already made substantial investments in search within the Lucene ecosystem. While most organizations do not directly use the open-source Lucene search library in production, the search application landscape is dominated by platforms that are built on top of Lucene such as Elasticsearch, OpenSearch, and Solr. For example, Elastic, the publicly traded company behind Elasticsearch, reports approximately 20,000 subscriptions to its cloud service as of Q4 FY2023.[1] Similarly, in the category of search engines, Lucene dominates DB-Engines Ranking, a site that tracks the popularity of various database management systems.[2] There's a paucity of concrete usage data, but it would not be an exaggeration to say that Lucene has an immense install base.

The most recent major release of Lucene (version 9), dating back to December 2021, includes HNSW indexing and search capabilities, which have steadily improved over the past couple of years. This means that differences in capabilities between Lucene and dedicated vector stores are primarily in terms of performance, not the availability of critical features. Thus, from a simple cost–benefit calculus, it is not clear that vector search requires introducing a dedicated vector store into an already complex enterprise "AI stack". Our thesis: Lucene is all you need.

We empirically demonstrate our claims on the MS MARCO passage ranking test collection, a standard benchmark dataset used by researchers today. We have encoded the entire corpus using OpenAI's `ada2` embedding endpoint, and then indexed the dense vectors with Lucene. Experimental results show that this combination achieves effectiveness comparable to the state of the art on the development queries as well as queries from the TREC 2019 and 2020 Deep Learning Tracks.

Our experiments are conducted with Anserini [29], a Lucene-based IR toolkit that aims to support reproducible information retrieval research. By building on Lucene, Anserini aims to bridge the gap between academic information retrieval research and the practice of building real-world search applications. Devins et al. [6] showed that capabilities implemented by researchers in Anserini using Lucene can be straightforwardly translated into Elasticsearch (or any other platform in the Lucene ecosystem), thus simplifying the path from prototypes to production deployments.

Our demonstration further shows the ease with which state-of-the-art vector search can be implemented by simply "plugging together" readily available components. In the context of the bi-encoder architecture, Lin [13] identified the logical scoring model and the physical retrieval model as distinct conceptual components. In our experiments, the logical scoring model maps to the OpenAI embedding API—whose operations are no different from any other API endpoint. What Lin calls the physical retrieval model focuses on the top-k retrieval capability, which is handled by Lucene. In Anserini, vector indexing and search is exposed in a manner that is analogous to indexing and retrieval using bag-of-words models such as BM25. Thus, the implementation of the state of the art in vector search using generative AI does not require any AI-specific implementations, which increases the accessibility of these technologies to a wider audience.

[1] https://ir.elastic.co/news/press-releases/default.aspx
[2] https://db-engines.com/en/ranking/search+engine

3 EXPERIMENTS

Experiments in this paper used the MS MARCO passage ranking test collection [2], which is built on a corpus comprising approximately 8.8 million passages extracted from the web. Note that since the embedding vectors are generated by OpenAI's API endpoint, no model training was performed. We used the 6980 development (dev) queries as well as queries from the TREC 2019 and 2020 Deep Learning (DL) Tracks.

Our experiments used the OpenAI `ada2` model [24] for generating both query and passage embeddings. This model is characterized by an input limit of 8191 tokens and an output embedding size of 1536 dimensions. However, to maintain consistency with the existing literature [20, 26], we truncated all passages in the corpus to 512 tokens. It is unknown whether OpenAI leveraged the MS MARCO passage corpus during model development, but in general, accounting for data leakage is challenging for large models, especially those from OpenAI that lack transparency.

Using `tiktoken`, OpenAI's official tokenizer, we computed the average token count per passage in our corpus to be 75.2, resulting in a total of approximately 660 million tokens. In order to generate the embeddings efficiently, we queried the API in parallel while respecting the rate limit of 3500 calls per minute. We had to incorporate logic for error handling in our code, given the high-volume nature of our API calls. Ultimately, we were able to encode both the corpus and the queries, the latter of which were negligible in comparison, in a span of two days.

As previously mentioned, all our retrieval experiments were conducted with the Anserini IR toolkit [29]. The primary advantage of Anserini is that it provides direct access to underlying Lucene features in a "researcher-friendly" manner that better comports with modern evaluation workflows, supporting easy reproducibility. Our experiments were performed with Anserini 0.23.0, which is based on Lucene 9.8.0. We used a Mac Studio with an M1 Ultra processor containing 20 cores (16 performance, 4 efficiency) and 128 GB memory, running macOS Sonoma 14.1.2 and OpenJDK 11.0.13. To better isolate retrieval performance, all our experiments were performed with pre-encoded (cached) query vectors. That is, we did *not* call the embedding API at query time.

Effectiveness results are shown in Table 1, where we report ranking quality in terms of standard metrics: reciprocal rank at 10 (RR@10), nDCG at a rank cutoff of 10 (nDCG@10), and recall at a rank cutoff of 1000 (R@1k). The effectiveness of the `ada2` embeddings is shown in the last row of the table. Note that due to the non-deterministic nature of HNSW indexing, effectiveness figures may vary slightly from run to run.

For comparison, we present results from a few select points of reference, classified according to the taxonomy proposed by Lin [13]; OpenAI's embedding models belong in the class of learned dense representations. Notable omissions in the results table include the following: the original OpenAI paper that describes the embedding model [24] does not report comparable results; neither does Izacard et al. [9] for Contriever, another popular learned dense representation model. Recently, Kamalloo et al. [11] also evaluated OpenAI's `ada2` embeddings, but they did not examine any of the test collections we do here. Looking at the results table, our main point is that we can achieve effectiveness comparable to the state of the art using a production-grade, completely off-the-shelf embedding API coupled with Lucene for indexing and retrieval.

	dev		DL19		DL20	
	RR@10	R@1k	nDCG@10	R@1k	nDCG@10	R@1k
Unsupervised Sparse Representations						
BM25 [20]	0.184	0.853	0.506	0.750	0.480	0.786
BM25+RM3 [20]	0.157	0.861	0.522	0.814	0.490	0.824
Learned Sparse Representations						
uniCOIL [20]	0.352	0.958	0.702	0.829	0.675	0.843
SPLADE++ ED [7]	0.383	0.983	0.731	0.873	0.720	0.900
Learned Dense Representations						
TAS-B [8]	0.344	0.977	0.721	0.841	0.685	0.873
TCT-ColBERTv2 [16]	0.358	0.970	0.720	0.826	0.688	0.843
Aggretriever [17]	0.362	0.974	0.684	0.808	0.697	0.856
OpenAI ada2	0.343	0.984	0.704	0.863	0.676	0.871

Table 1: Effectiveness of OpenAI `ada2` embeddings on the MS MARCO dev queries and queries from DL19/DL20, compared to a selection of other models. All results are from Pyserini's two-click reproductions [14], which may differ slightly from the original papers.

Finally, we provide performance figures running on 16 threads. With the default Anserini indexing configuration—the parameter `M` set to 16 and `efC` set to 100, without final segment optimization—we were able to achieve around 22 queries per second with `efSearch` set to 1000 on the MS MARCO development queries, retrieving 1000 hits per query. The resulting index occupies 51 GB (with the `du -h` command) with 25 index segments. Keep everything exactly the same, but merging the index down into a single segment, we are able to achieve around 395 queries per second. We note that although this optimization greatly increases search performance, it is only applicable for static collections.

4 DISCUSSION

Our demonstration shows that it is possible today to build a vector search prototype using OpenAI embeddings directly with Lucene. Nevertheless, there are a number of issues worth discussing, which we cover below.

Pace of development. We concede that Lucene has been a relative laggard in the development of support for dense retrieval, at least compared to standalone vector stores. Despite this, we believe that recent announcements point to substantial and sustained investments in the Lucene ecosystem moving forward. For example, in its Q4 FY 2023 report, Elastic announced the Elasticsearch Relevance Engine, "powered by built-in vector search and transformer models, designed specifically to bring the power of AI innovation to proprietary enterprise data." A recent blog post[3] from Amazon Web Services explained vector database capabilities in OpenSearch, providing many details and reference architectures. These are just two examples of industry commitments that help bolster the case for Lucene that we have articulated here. Already, Lucene has released version 9.9.1 (which we did not examine in this work), with several new features such as int8 vector quantization. Overall, we are optimistic about the future of the ecosystem.

[3] https://aws.amazon.com/blogs/big-data/amazon-opensearch-services-vector-database-capabilities-explained/

Performance. Ma et al. [21] recently benchmarked Lucene 9.5.0 against Faiss [10] and concluded that Lucene still lags in terms of indexing speed, query latency and throughput, and related metrics. Those experiments suggest that Lucene achieves only around half the query throughput of Faiss under comparable settings, but appears to scale better when using multiple threads. The results, however, only capture a snapshot in time; here we report results using Lucene 9.8.0, and Lucene 9.9.1 is already available. Nevertheless, it would be fair to say that there remains a performance gap between Lucene and Faiss, at least along some dimensions of interest. However, the latter is relatively mature and hence its headroom for performance improvements is more limited. In contrast, we see many more opportunities for gains in Lucene. Coupled with signs of strong commitment from industry players (discussed above), we believe that the performance gap between Lucene and Faiss (as well as other dedicated vector stores) will decrease over time.

Alternatives. We acknowledge a number of competing alternatives that deserve consideration. Note that the core argument we forward is about cost–benefit tradeoffs: In our view, it is not clear that the benefits offered by a dedicated vector store outweigh the increased architectural complexity of introducing a new software component within an enterprise. From this perspective, we can identify two potentially appealing alternatives:

- *Fully managed services.* One simple way to reduce architectural complexity is to make it someone else's problem. Vespa[4] is perhaps the best example of this solution, providing both dense retrieval and sparse retrieval capabilities in a fully managed environment, eliminating the need for users to explicitly worry about implementation details involving inverted indexes, HNSW indexes, etc. Vepsa provides a query language that supports a combination of vector search, full-text search, as well as search over structured data. Our main question here concerns traction and adoption: as a brownfield application, we're not convinced that enterprises will make the (single, large) leap from an existing solution to a fully managed service.
- *Vector search capabilities in relational databases.* In the same way that vector search grows naturally out of an already deployed and mature text search platform (e.g., Elasticsearch), we can see similar arguments being made from the perspective of relational databases. Despite numerous attempts (spanning decades) at toppling its lofty perch [25, 27], relational databases remain a permanent fixture in enterprise "data stacks". This means that by building vector search capabilities into relational databases, enterprises gain entrée into the world of dense retrieval (essentially) for free. A great example of this approach is pgvector,[5] which provides open-source vector similarity search for Postgres. We find the case compelling: if your enterprise is already running Postgres, pgvector adds vector search capabilities with minimal additional complexity. It's basically a free lunch.

5 CONCLUSIONS

There is no doubt that manipulation of dense vectors forms an important component of search today. The central debate we tackle is how these capabilities should be implemented and deployed in production systems. The dominant narrative is that you need a new, distinct addition to your enterprise "AI stack"—a vector store. The alternative we propose is to say: If you've built search applications already, chances are you're already invested in the Lucene ecosystem. In this case, Lucene is all you need. Of course, time will tell who's right.

[4] https://vespa.ai/
[5] https://github.com/pgvector/pgvector

ACKNOWLEDGMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. We'd like to thank Josh McGrath and the team at Distyl for providing support to access OpenAI APIs.

REFERENCES

[1] A. Asai et al. 2023. Retrieval-based Language Models and Applications. In *ACL Tutorials*.
[2] P. Bajaj et al. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3* (2018).
[3] Haonan Chen et al. 2023. End-to-End Retrieval with Learned Dense and Sparse Representations Using Lucene. *arXiv:2311.18503* (2023).
[4] N. Craswell et al. 2019. Overview of the TREC 2019 Deep Learning Track. In *TREC*.
[5] N. Craswell et al. 2020. Overview of the TREC 2020 Deep Learning Track. In *TREC*.
[6] J. Devins et al. 2022. Aligning the Research and Practice of Building Search Applications: Elasticsearch and Pyserini. In *WSDM*.
[7] T. Formal et al. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *SIGIR*. 2353–2359.
[8] S. Hofstätter et al. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *SIGIR*.
[9] G. Izacard et al. 2021. Towards Unsupervised Dense Information Retrieval with Contrastive Learning. *arXiv:2112.09118* (2021).
[10] J. Johnson et al. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[11] E. Kamalloo et al. 2023. Evaluating Embedding APIs for Information Retrieval. In *ACL Industry Track*.
[12] V. Karpukhin et al. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*.
[13] J. Lin. 2021. A Proposed Conceptual Framework for a Representational Approach to Information Retrieval. *arXiv:2110.01529* (2021).
[14] J. Lin. 2022. Building a Culture of Reproducibility in Academic Research. *arXiv:2212.13534* (2022).
[15] J. Lin et al. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *SIGIR*.
[16] S. Lin et al. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *RepL4NLP Workshop*.
[17] S. Lin et al. 2023. Aggretriever: A Simple Approach to Aggregate Textual Representations for Robust Dense Passage Retrieval. *TACL* 11 (2023), 436–452.
[18] S. Lin and J. Lin. 2023. A Dense Representation Framework for Lexical and Semantic Matching. *ACM Transactions on Information Systems* 41 (2023). Issue 4.
[19] X. Ma et al. 2022. Another Look at DPR: Reproduction of Training and Replication of Retrieval. In *ECIR*.
[20] X. Ma et al. 2022. Document Expansions and Learned Sparse Lexical Representations for MS MARCO V1 and V2. In *SIGIR*.
[21] X. Ma et al. 2023. Anserini Gets Dense Retrieval: Integration of Lucene's HNSW Indexes. In *CIKM*.
[22] Y. Malkov and D. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.
[23] G. Mialon et al. 2023. Augmented Language Models: a Survey. *arXiv:2302.07842* (2023).
[24] A. Neelakantan et al. 2022. Text and Code Embeddings by Contrastive Pre-Training. *arXiv:2201.10005* (2022).
[25] A. Pavlo et al. 2009. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*.
[26] R. Pradeep et al. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. *arXiv:2101.05667* (2021).
[27] M. Stonebraker and J. M. Hellerstein. 2005. What Goes Around Comes Around.
[28] L. Xiong et al. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *ICLR*.
[29] P. Yang et al. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *Journal of Data and Information Quality* 10, 4 (2018).