Towards a More Scalable, Efficient, and User-Friendly Training Data Search Engine

Andrew Zheng **Rifaa Qadri** **Vrundal Shah** **Xinchen Yang** **Joseph Hong**

1 Introduction

The rapid advances in technology and proliferation of internet produced an overflow of information. In the realm of Natural Language Processing (NLP), tasks such as question answering, text summarizing, fact checking, machine translation among others, rely heavily on practical and efficient retrieval mechanisms to retrieve relevant documents pertaining to a user's query Thakur et al. 2021.

Piktus et al. 2023 recognized the urgent need to provide a fast and user-friendly mechanism for qualitative analysis of large scale textual corpora and thus created GAIA: a simple search engine giving relevance based interface to 4 popular, large-scale, textual datasets which rely to a big extent on data mined from Common Crawl. To match queries and documents, BM25 weighting was utilized, which extends TF-IDF, the term specificity retriever. However, such traditional techniques still limit us to near-exact matches to retrieve documents and suffer from the lexical gap and therefore do not generalize well. Izacard et al. 2022. By contrast, neural networks allow learning beyond lexical representations and capture semantic relationships.

Our main contribution would be a evaluating Pyserini's capabilities for sparse, dense vector and hybrid retrieval methods To this end, our work aims to create a scalable, efficient and user friendly zero-shot setup retrieval system, able to retrieve relevant documents for a given query without having been trained on any data pertaining to it. In addition, we will work towards creating a user friendly interface and performing surveys to examine the effectivenss of the search engine.
The code of this project will be managed in the GitHub repo here: `github.com/rifaaQ/cmsc674`

2 Research Questions

In this research project, we will answer the following question:
Can a better retrieval method build a better search engine and facilitate the qualitative analysis of NLP datasets? Given a query of various length, can we achieve high-quality, large scale retrieval using a variety of datasets? If time allows: Can a better search engine help identify the specific and different data sources that were used to train an LLM? Could this help assess the quality and accuracy of the LLM output? Can we further identify biases or data imbalances used to train the LLM?

3 Methods

All datasets will be sourced from Common Crawl. We will begin by exploring a few tokenization techniques from HuggingFace. This process typically includes "removing stop words, stemming, lemmatization, and removing non-alphanumeric characters" Piktus et al. 2023 In particular, subword tokenization has shown promise in IR and appears to be a good candidate. To build the index, we will rely on the capabilities of the Pyserini library's `https://pypi.org/project/pyserini/` which facilitates for both sparse and dense vector retrieval, as well as hybrid approaches. We note that this is a significant different than the GAIA search which focuses solely on sparse retrieval using BM25 indexes. Once the index is ready, we will host it and serve the search functionality to the client. We understand the importance of maintaining the interoperability between Pyserini and Huggingface when building and deploying an interactive search application for textual datasets.

The evaluation of various information retrieval systems is heavily dependent on the retrieval algorithm serving the search indices being evaluated. For this reason, we will focus our research on find-

ing a novel method to perform the vector search. We are interested in exploring contrastive learning methods for dense information retrieval. The Contriever architecture begins by utilizing a transformer network to embed both queries and documents independently. We are interested in discovering what improvement this would lead to in the context of robustness. After applying the encoder, the relevance score between a query and a document is given by the dot product between their representations as follows:

$$s(q,d) =< f_\theta(q), f_\theta(d) >$$

Next, the representation for a query resp document is obtained by averaging the hidden representations of the last layer.

The training is performed by the utilization of the contrastive InfoNCE loss, defined as:

$$L(q,k_+) = \frac{-\exp(s(q,d)/\tau)}{\exp(s(q,d)/\tau) + \sum_i \exp(s(q,d)/\tau)}$$

where τ is a temperature parameters. Izacard et al. 2022 This process is repeated until Contriever learns to distinguish between positive and negative pairs with high accuracy. We can see this as an optimization problem as we encourage positive pairs to have high scores and negative pairs to have low scores. Once trained, we can use it to retrieve relevant documents for any given query.

4 Literature Review

4.1 Retrieval Methods

Information retrieval methods have been periodically benchmarked to evaluate their effectiveness. For example, Thakur introduced an evaluation benchmark for information retrieval named Benchmarking-IR, which consists of 18 public datasets from heterogeneous domains. Thakur evaluated 10 retrieval systems on the proposed benchmark, showing that BM25 is a robust system with high performance but also high cost, while dense and sparse-retrieval models cost less computational resources but achieve worse performance Thakur et al. 2021.

For working with a large database that includes terabytes of data, it is important to have a method that could query a database fast. BM-25, although a good method for producing accurate query results, is not very good at including contextual data from a query nor being efficient for database standards.

One promising area to explore is a vectorized search where each query and document is encoded into a vector containing contextual data within it. With the embedding of queries and documents into vectors, information retrieval becomes much faster. Inspiration can be taken from Large Language Model Algorithms which seek not only to find the top k most relevant document to a query but also to generate a response to a specific query.

A compelling starting point is outlined in the ColBERT research paper which introduces a Large Language Model that seeks to introduce the idea of creating separate embeddings for the queries and documents separately before making MaxSim calculations between the documents and the queries. During the embedding process, an attention mechanism is used to consider the context of query as well as the contexts in the Document. Ideas similar to ColBERT can be generalized towards databases for the creation of a user-friendly interface with an effective backend for querying a large database of document data.

Another state-of-the-art embedding technique worth considering is a Contrastive Learning approach which introduces an unsupervised learning approach for zero-shot learning. Similar to ColBERT, separate embeddings for queries and documents were introduced. Contrastive Learning then trains these dense retrievers by comparing to the documents to see which document yields the smallest loss based on a loss function mentioned in the paper. Ideas mentioned with unsupervised contrastive learning can be applied to speed up the database queries dense retrievers have less dimensions and maintain all queries and documents in a vectorized form.

4.2 Embedding

Embedding methods have also been benchmarked by researchers. For example, Muennighoff presented the Massive Text Embedding Benchmark (MTEB), which includes 8 text embedding tasks covering 58 datasets. Through MTEB, Muennighoff benchedmarked 33 state-of-the-art embedding models and figured out that there is no embeddding method that dominates across all tasks Muennighoff et al. 2023. In particular for the retrieval

task, SGPT-5.8B-msmarco is the best embedding model on the BEIR subset in MTEB as well as on the full BEIR benchmark

4.3 User-Friendly Search Interfaces

Previously, toolkits have been created to allow users to interact and compare state-of-the-art methods in Large Language Models. One older toolkits that was aimed towards document relevance ranking was Anserini, a paper that sought to do a combination ranking documents accurately and also provide certain standardized testing options for one method through various means such as comparison to other methods or comparison to other datasets.

Recently, other toolkits have also emerged. Pyserini is a python toolkit that was created that focused on Information Retrieval in the form of document ranking. More specifically, Pyserini compares ranking methods with each other across multiple datasets to allow the best user interaction for comparing their own methods or to interact with certain aspects of the provided ranking methods.

Finally, a toolkit that was presented in 2023 is GAIA, a user-friendly interface that seeks to search for documents across a couple of databases. GAIA sought to build on top of Pyserini by adding more datasets which can be used for Large Language Model research and to build a user-friendly interface that made it easy to compare document-ranking ideas introduced in Pyserini with the multitude of datasets they provided. Similar to Pyserini, GAIA incorporates an option for a BM-25 search that can be used for ranking documents.

However, GAIA has lots of faults. In this research project, we seek to improve the user-interface the interface is poorly built and not easy to navigate. In addition, we will improve the searching method because BM-25 does not take into account any contextual data and is very slow compared to large-corpus data standards.

5 Project Outline

This project will be done in Python. We will need access to the Common Crawl data stored AWS, as well as a cloud database able to support TB-scale corpora. To this extent, we will contact datastax.com for help.

Work Distribution: Everyone will contribute to the analysis.

5.1 Timeline

1. (Sep 22 - Oct 3): Make a dataset of positive and negative pairs(maybe a few dozens of them) Survey how many methods to compare and choose from for embeddings and retrieval

2. (Oct 4 - Oct 13): Setup methodology and pipeline to test different methods out and how they perform on the pairs developed.

3. (Oct 14 - Oct 20): Analysis of different methods to decide which embedding method to use.

4. (Oct 21 - Oct 28): which retrieval method to use?

5. (Oct 29 - Nov 4): Set up AWS account and Grab common crawl, and/or c4, ...(UMD has some of these). Initial steps to make the databases ready breaking it up, hashing, etc. Implement the corresponding chosen methods

6. (Nov 11 - Nov 19): Stash the resulting data into a datastax database. So you can search it by vector

7. (Nov 19 - Nov 29): MVP for frontend integrated with the backend (can use Socket.io here)

8. (Nov 30 - end of semester): Buffer for deployment issues or unforeseen circumstances during the timeline of the project

5.2 Deliverables

For this project, we will create an web-search application that will allow someone to input a query and give that person which documents it came from in Common Crawl. When a query is entered, the application will embed the text using the preferred embedding method we find. Then it will access the vector embeddings of the documents from Common Crawl, which are stored in the DataStax database, and finds the most closely related documents. If possible, links to the documents and the dataset (e.g., C4) from which it came from will also be provided to the user, and the user will be able to

change how many of the closest related documents they want to see.

Our application will be an improvement of the already existing GAIA search at huggingface.co/spaces/spacerini/gaia

5.3 Validation Methods

When getting the data from Common Crawl we will create tests to ensure that data has been retrieved successfully. We will test this by retrieving a small portion of a some crawl, verify that we actually get the data and the format of the data is correct to be used in our embedding test, then we move on to retrieve larger portions of data.

For our text-s we will make sure the embedding model loads correctly and that there and also include tests that will assess the loading and retrieval.

In our web-application search engine we will create tests for the input query the user inputs. We make sure to sanitize the input and make sure the input is in the correct format and doesn't break anything.

References

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2023. Mteb: Massive text embedding benchmark.

Aleksandra Piktus, Odunayo Ogundepo, Christopher Akiki, Akintunde Oladipo, Xinyu Zhang, Hailey Schoelkopf, Stella Biderman, Martin Potthast, and Jimmy Lin. 2023. Gaia search: Hugging face and pyserini interoperability for nlp training data exploration.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models.