Exploring Dense Retrieval via Approximate Nearest Neighbors

Ashwin Guptha (asg8739), Geetika Bandlamudi (gb2642)

Link to the code repo: WebSearchEnginesProject

Abstract

This project addresses the limitations of sparse lexical retrieval methods, such as BM25, by leveraging dense retrieval techniques based on embeddings generated from MiniLM. The focus is on finding the nearest passages in a given set efficiently. The proposed system incorporates a multi-faceted approach, involving the transformation of the MS MARCO passage dataset into a high-dimensional space using MiniLM, fast approximate nearest neighbor algorithms (HNSW), proper evaluation methodologies, and a potential hybridization with traditional methods like BM25. The system integrates a ladder model that combines BM25 for keyword-based relevance and semantic similarity through MiniLM embeddings. This conjunctive approach aims to enhance precision by selecting passages enhanced by both methods. We have added examples to share our view and take over the performance and results of these methods.

Problem Statement

Sparse lexical retrieval methods such as BM25 miss a lot of good results due to vocabulary mismatch and related issues. One approach to overcome this issue uses query expansion. More recently, it has been proposed to instead map queries and documents into a high-dimensional space using say a BERT transformer, and to then retrieve the documents that are closest to the query in this high-dimensional space, Finding these nearest neighbors could then be done via optimized Approximate Nearest Neighbor (ANN) methods such as HNSW, DiskANN, or NSG. This project has several different parts that are needed: (1) Transforming a collection such as MSMarco and the queries into the high-dimensional space using (say) BERT, (2) a fast ANN algorithm such as HNSW, (3) possibly also combining such a dense retrieval with traditional methods such as BM25 into a hybrid method.

Dataset

MS MARCO [1] (MicroSoft MAchine Reading COmprehension) is a large-scale dataset focused on machine reading comprehension. Since its initial release, benchmarking efforts for several NLP and IR tasks have made use of this dataset—including. question-answering, passage ranking, document ranking, keyphrase extraction, and conversational search. We used the passage dataset for this project. We have chosen the passage dataset to avoid the additional parsing needed for the document dataset. This provides us an opportunity to directly work with the webpage content. This passage dataset is based on the public MS MARCO dataset, although our evaluation will be quite different. Our dataset is a little over 2.9 GB with 8,841,823 entries in a TSV format with the pid, passage where pid is the passage ID.

Link to Dataset: https://microsoft.github.io/msmarco/Datasets.html

Overview

In the next few sections, we will discuss the various approaches we used and the performance and accuracy of those methods. We start with discussing the Embedding Generation phase, and how and why we chose MiniLM for this part. Next, we discuss the various Similarity Search algorithms like Cosine Similarity, Nearest Neighbours algorithm, and Semantic Search and judge the performance of each of these. We later delve deep into the Approximate Nearest Neighbours algorithm - HNSW and the intricate details about the working of the algorithm for our use case. Then, we propose a new ladder approach that combines the benefits of BM25 with the best algorithms seen so far. As a next step, we also discuss our attempt to enhance the quality of the results through query expansion. Finally, we discuss our take using a few examples. Just a heads up, we have used documents and passaged interchangeably throughout this report. Let's get started!

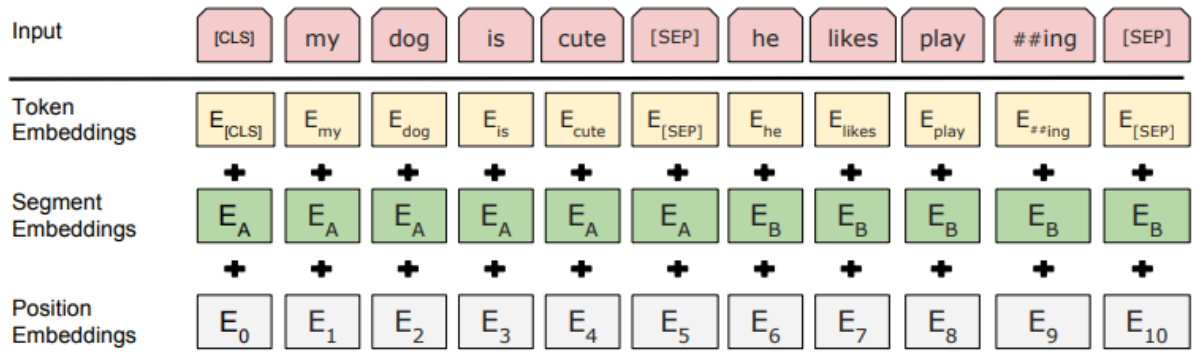
Embedding Generation

Embedding generation in machine learning and web search engines is a critical concept that encompasses the transformation of high-dimensional data into a lower-dimensional space, facilitating various tasks such as improving the efficiency and accuracy of search algorithms. In machine learning, embeddings are dense vector representations of data where the distance between vectors is indicative of the similarity between the entities they represent. The primary

goal of embeddings is to capture the essence of data in a format that is more manageable and useful for machine learning models. Embeddings in web search engines are pivotal in understanding the intent behind queries and the content of web pages. By converting words, phrases, and entire documents into vector representations, search engines can more effectively match queries with relevant results. These models understand the context and meaning of words in queries and documents, going beyond simple keyword matching.

BERT

BERT [2] (Bidirectional Encoder Representations from Transformers) is a bidirectional transformer pre-trained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia. BERT is built on the Transformer architecture, which relies on attention mechanisms. Unlike traditional sequence-based models that read text input sequentially, the Transformer processes the entire text at once, allowing it to capture more complex word relationships. Traditional language models were either left-to-right or right-to-left, limiting their understanding of context. BERT overcomes this by using a "masked language model" (MLM) approach, where random words in a sentence are masked out and the model is trained to predict them, considering both left and right context.



BERT tokenizes passages into words and subwords (to handle unknown words), and special tokens are added:

> [CLS]: A special token placed at the beginning of the text used for classification tasks. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.

[SEP]: A special token that separates segments. In this example, it separates two sentences or segments: "my dog is cute" and "he likes playing".

As shown in the Figure above, each token is converted into a vector using an embedding matrix called a token embedding. Segment embeddings are used to differentiate between different segments or sentences. In this example, E_A is used for the first sentence and E_B for the second. This allows the model to process paired sentences. Positional embeddings provide information about the token positions within the sequence. Since the Transformer does not have a recurrent structure and does not process the tokens sequentially, it requires positional encodings to understand the order of tokens. The combination of these three embeddings gives a detailed representation of each token, accounting for its meaning, its role within its sentence, and its position in the sequence. The sum of these embeddings is then passed through the Transformer's multi-layered bidirectional architecture, which processes the input in a context-aware manner, leading to highly contextualized token representations.

Why MiniLM?

MiniLM [4] is a variant of the transformer architecture, similar to BERT, but with modifications to make it more computationally efficient while maintaining strong performance in various natural language processing (NLP) tasks. The preference for MiniLM embeddings on the entire passage over a standard Word2Vec for each word in the MSMARCO passage dataset can be attributed to several key factors rooted in the inherent architectural differences. MSMARCO, a large-scale dataset designed for information retrieval and question-answering tasks, presents unique challenges that MiniLM is particularly well-equipped to handle.

Word2Vec generates embeddings based on local window-based contexts, meaning each word is represented by a single vector regardless of its usage in different contexts. This approach fails to capture the nuances of words with multiple meanings. MiniLM takes care of this and words with multiple meanings and usages are embedded differently based on context, leading to a more accurate representation of the language which in turn, is more useful for a search engine. The task in MSMARCO often involves matching user queries with relevant passages. MiniLM’s ability to understand the semantic meaning of sentences as a whole allows for more effective matching of queries to passages, especially in cases where keyword overlap is minimal.

Similarity Search Algorithms

Similarity search in web search engines refers to the process of retrieving documents or items that are similar to a given query or document. We have seen the process of using traditional keyword-based search where the focus is on matching exact words or phrases in assignments 2 and 3. But here the similarity search aims to find items that are conceptually or contextually similar to the input. There are multiple ways of doing this and we have chosen a few of the approaches that we found to be common in the industry for such use cases previously.

Cosine Similarity Search

This is a metric used to measure the cosine of the angle between two vectors. In information retrieval, it is often employed to calculate the similarity between the vector representations of documents. Here, since we are working with passages, we represent the passages in a higher dimensional space using MiniLM as explained above. The search function takes the query as the input, encodes it into the higher dimension using MiniLM, and calculates the cosine similarity with each document's vector representation. The formula for cosine similarity (cos θ) between two vectors A and B is given by:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

In the context of information retrieval using MiniLM embeddings, the vectors A and B correspond to the vector representations of the query and the document, respectively. The dot product measures the similarity in direction between the vectors, and the denominator normalizes the result based on the lengths of the vectors.

The cosine similarity ranges from -1 to 1, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect dissimilarity. In the context of search engines, a higher cosine

similarity score between the query and a document implies a greater degree of similarity, helping the search engine rank and retrieve documents that are more contextually relevant to the user's query. We have used the sentence transformer util library to perform the cosine similarity search. It takes the query embedding and the passage corpus embeddings to find the cosine similarity. We later use the top_k() method to find the most similar documents by similarity score.

We noticed that the cosine similarity method is extremely slow given that the algorithm calculates the cosine angle for every pair of passages to find the most similar passages. This is very general and needs more improvement. Nowadays, this algorithm is often part of a larger and faster algorithm that uses cosine as a measure of distance between items in the higher dimensional space.

K Nearest Neighbors

K Nearest Neighbors (KNN) is a very widely applied algorithm in machine learning for various tasks, including similarity search. In our context, KNN provides a way for us to identify documents that closely match a given query. KNN transforms both the passages and queries into a higher dimensional space facilitating the comparison of their positions. This algorithm is influenced by the sparsity of the data, and in high-dimensional spaces, distances tend to become less meaningful which is called the curse of dimensionality. We wanted to know if the overall geometric arrangement of vectors in the feature space is significant for this use case.

Nearest Neighbour — Distance Measures

- Given two feature vectors with numeric values

$$A = (a_1, a_2, ..., a_n) \text{ and } B = (b_1, b_2, ..., b_n)$$

- Use the *distance measure*:

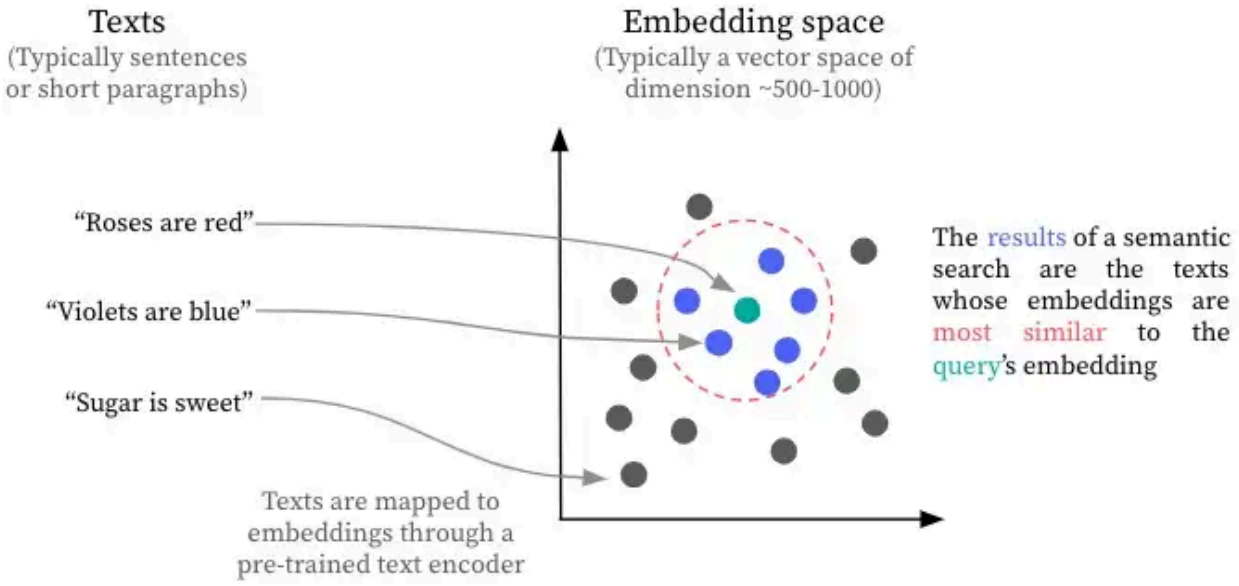
$$d = \sqrt{\sum_{i=1}^{n} \frac{(a_i - b_i)^2}{R_i^2}} = \sqrt{\frac{(a_1 - b_1)^2}{R_1^2} + \frac{(a_2 - b_2)^2}{R_2^2} + ... + \frac{(a_n - b_n)^2}{R_n^2}}$$

R_i is the *range* of the ith component

Our system employs sci-kit-learn's Nearest Neighbors with the Euclidean metric to find the most similar documents in a corpus. The algorithm calculates the Euclidean distance between the query and each document, identifies the K nearest neighbors, and returns them as search results. We observed that the KNN algorithm performs just as well as the cosine similarity search. We wanted to explore faster algorithms that take the semantic information also into consideration while calculating the similarity. Naturally, the next choice was semantic search.

Semantic Search

In the process of harnessing machine understanding capabilities, we explored more about leveraging natural language understanding and semantic analysis to comprehend the meaning behind user queries and the content of documents. This approach [5] aims to enhance the precision and relevance of search results by considering the context, intent, and semantics of the information. Semantic search generates dense vector representations for both queries and documents using MiniLM just like how it is described above. These embeddings capture intricate semantic relationships, allowing for a nuanced understanding of textual data. Semantic search is especially valuable when dealing with ambiguous queries, capturing nuanced semantic relationships, and improving the overall precision of information retrieval.



As we know, the semantic search utility function from the sentence transformer library uses the cosine similarity operation. So a natural question would be as to why we have tried both

approaches if all we are interested in is the cosine similarity between the query and the documents. Cosine similarity is a mathematical metric that measures similarity between two vectors, whereas semantic search is an information retrieval approach that focuses on understanding the semantics of text for more contextually relevant results. Semantic search often involves the use of cosine similarity as part of its calculations, but it encompasses a broader set of processes beyond simple vector similarity.

Our experimentation involved comparing the performance of both approaches—direct cosine similarity and semantic search. We observed that while cosine similarity provided effective results in capturing surface-level similarities, semantic search outperformed in scenarios where a deeper understanding of the underlying meaning was crucial. This was particularly evident in instances of ambiguous queries, where semantic search demonstrated superior performance by considering the broader context and semantic relationships within the documents. For example, we searched for “purple pride” and got results related to rainbows, along with LGBTQ+-related content. The nuanced understanding facilitated by semantic search contributed to more accurate and contextually relevant passage similarity based on our dataset. However, we are still suffering from the long duration it takes to perform a search. We explore the recent research algorithms that are being brought close to developers and scientists across various domains and projects - Approximate Nearest Neighbours. Spotify, Amazon, and Microsoft have conducted various studies in this domain and more recently, ElasticSearch from Amazon has incorporated this algorithm into its search techniques. Let us use one of those algorithms for our use case.

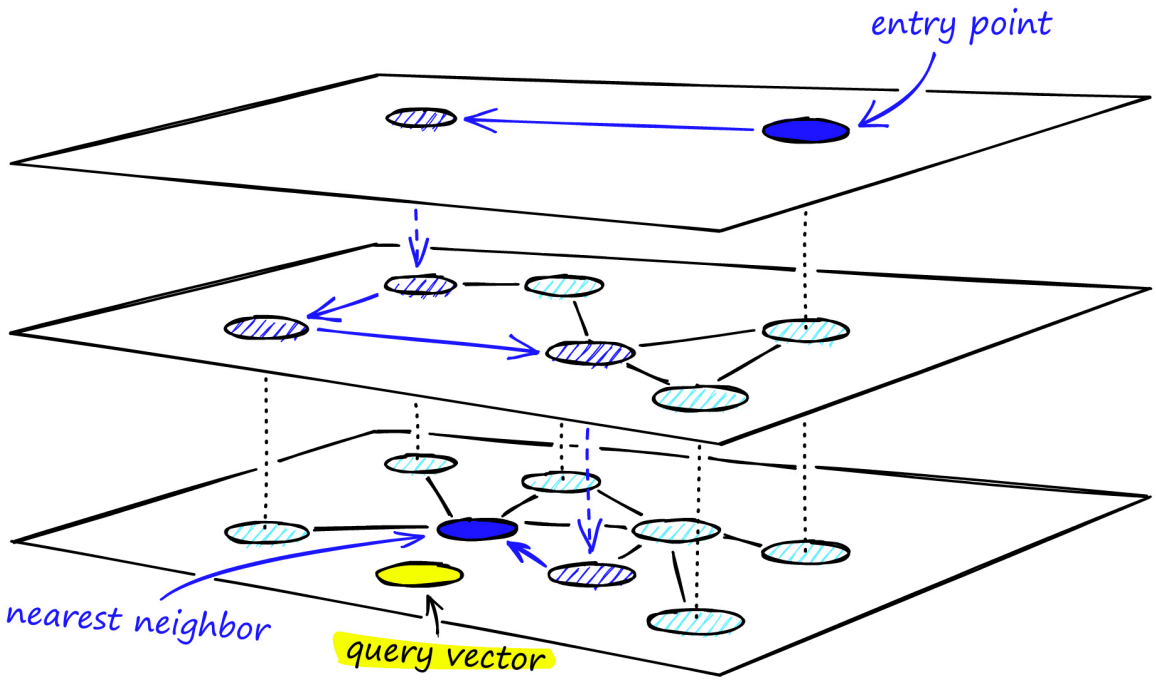
Approximate Nearest Neighbours Model

The nearest neighbor problem is a problem in the algorithms space where, given a set of points in a metric space, the goal is to construct a data structure that would efficiently identify, for any given point, its closest neighbor from the initial set without computing all distances between the points. To make things faster and use less memory, we sometimes play the "approximate" nearest neighbor game. It's like saying, "I don't need the absolute closest buddy, just someone pretty close will do." This is perfect for our use case here because eventually, we are leaning towards finding the closest documents to a set of documents that we shortlist by other algorithms. Also, we are interested in an “approximation” to find similar passages for a given query because we're dealing with a large dataset, and computing exact distances between each document and query could be computationally expensive and memory-intensive.

HNSW

Hierarchical Navigable Small World (HNSW) is a state-of-the-art algorithm used for an approximate search of nearest neighbors. Under the hood, HNSW constructs optimized graph structures making it very different from other approaches that were discussed in previous parts of this report. Its structure represents a multi-layered graph with fewer connections on the top layers and more dense regions on the bottom layers. The search starts from the highest layer and proceeds to one level below every time the local nearest neighbor is greedily found among the layer nodes. Ultimately, the nearest neighbor on the lowest layer is the answer to the query. One of the most important and best parts of this Disk-based ANN is the time complexity which is O(log n).

One of the major reasons why we chose HNSW [6] over other methods is its extreme efficiency. When working with higher dimensional vectors, we observed that the previous methods were taking a lot of time to find the nearest neighbors and we needed a faster algorithm. Moreover, we wanted to avoid the curse of dimensionality in our use case. As we have discussed before, traditional methods struggle with the curse of dimensionality, where the distance between points becomes less meaningful as the number of dimensions increases. HNSW's hierarchical structure helps mitigate this issue. HNSW provides an approximate nearest neighbor search with guarantees. While it sacrifices exactness for efficiency, the algorithm ensures that the identified nearest neighbors are close to the true neighbors. Since in our use case, we are not going in search of “exactness”, but rather in search of “correctness”, we have chosen this algorithm.



One challenge we faced was loading and working with a very large corpus embedding. Hence, we split it into multiple chunks and worked with them individually. We initialized the HNSW vector space as 8Mx384 since we had 8M corpus embeddings each with a dimension of 384. Taking advantage of the utility functions in the HNSW library, we added the embeddings to the HNSW index while saving the index along the way. We noticed that the denser the index, the more time it took for the embeddings to be added to the index. However, the overall process took a few hours.

BM25

The user query is preprocessed before being sent as input to BM25 in the ladder method. For this version of the algorithm, conjunctive queries are being used as disjunctive or even a hybrid version will lead to passages with no context to the query to be returned. If a query of length l does not have a common passage ID between the query words in BM25, then only the most important l-1 words of the query are taken into account. This repeats till the BM25 algorithm is able to return a set of passage IDs. The process of selecting the most important keywords from the query will be discussed in the next section.

The BM25 algorithm represents a pivotal component in the information retrieval system. To enhance the efficiency of the retrieval process, a merge and sort operation is integral after the initial computation of BM25 scores. This operation involves the consolidation of temporary files, which contain intermediate results, and their subsequent organization. The Unix sort command serves as the cornerstone for this operation, facilitating the merger of all temporary files. It

meticulously orders the entries first by the word and subsequently by the document ID, resulting in the creation of a unified index file named merge.index.
The Unix sort utility is invoked with a specified buffer size of 1GB, a measure adopted to modulate memory consumption during the sorting phase. This parameterization ensures that the sorting remains within acceptable memory bounds, thereby optimizing system performance. In addition to its sorting capabilities, the utility also excels in deduplication; it deftly identifies and removes duplicate entries from the sorted data, ensuring the uniqueness of each line in the resultant index. The following is the formula for calculating BM25 for each document:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)},$$

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

N: Total number of documents in the corpus
n(qi): Total number of documents that contain the term q
f(qi, D): Total number of times qi occurs in the document D.
b: 0.75
k: 1.5
Avgdl: Average document length
D: Total number of words in the document D

Keyword extraction

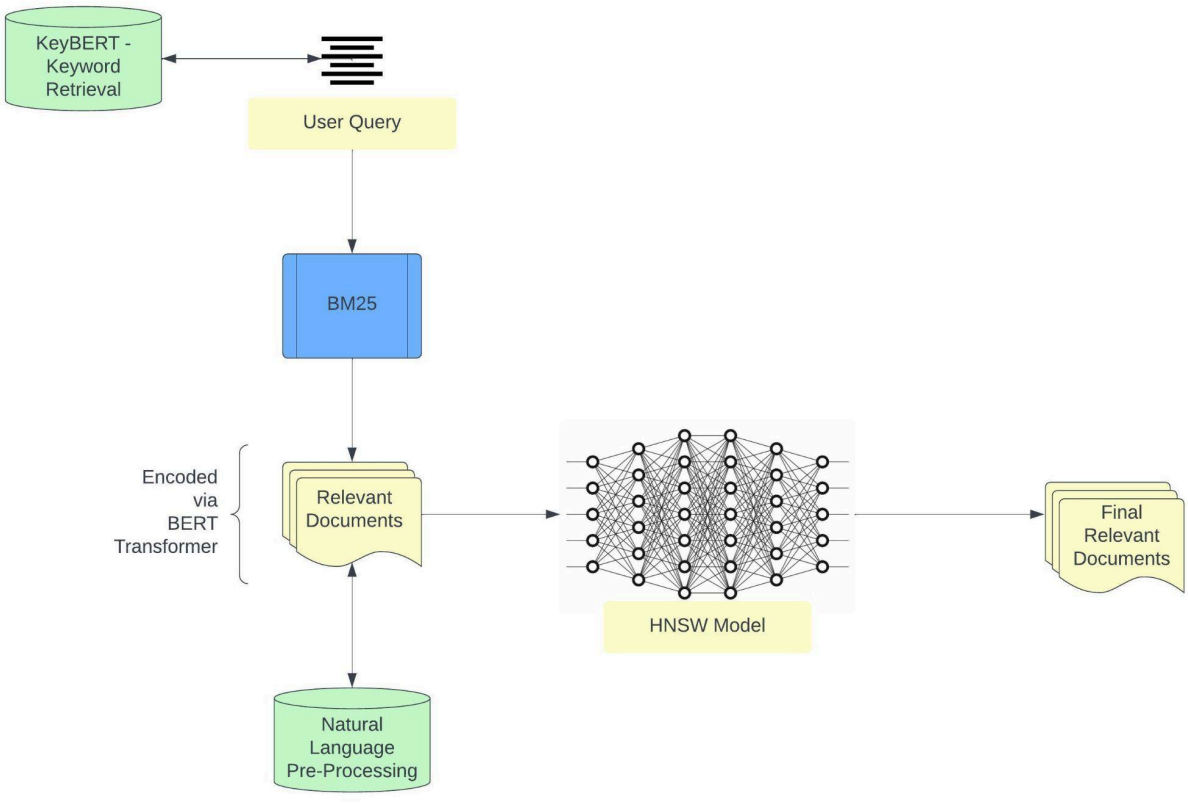
This process is used for query expansion as well as to get conjunctive query results in BM25 in case the original query cannot return any common passages.

For query expansion, a list of relevant passages is taken, usually after BM25 (in the ladder method), and combined into a single large string. This aggregation is done to create a comprehensive context for KeyBERT [9][10] to analyze, as it will look for keywords that are relevant across the entire set of passages. KeyBERT, a model that uses BERT embeddings for keyword extraction, is then utilized to extract keywords from the concatenated passages. The model is instructed to ignore the original query terms as stop words during this process, ensuring that the keywords extracted are additional terms and not those already specified by the user. Out of the extracted keywords, only the top k keywords are chosen. These are considered the most relevant or important terms related to the context provided by the list of passages. KeyBERT is effective because it uses BERT's deep understanding of context and semantics,

which allows it to capture more nuanced and meaningful keywords than methods that rely on frequency or statistical measures alone.

Ladder Method

From the above two methods, we realized that an effective search engine needs to combine the benefits of keyword-based relevancy and semantic similarity. Hence, we proposed a ladder approach that takes the input query and finds the conjunctive most relevant documents using BM25. The results of BM25 are encoded in a higher dimensional space to find the most similar passages. We have chosen the total passages returned to be 1000. A conjunctive approach is chosen to ensure that the final result set is highly relevant and satisfies both traditional keyword-based relevance (BM25) and semantic similarity (ANN) criteria. This approach aims to improve precision by requiring documents to meet criteria from both methods, reducing the likelihood of false positives and ensuring a more focused set of results. A rough architecture of this approach is shown in the figure below.



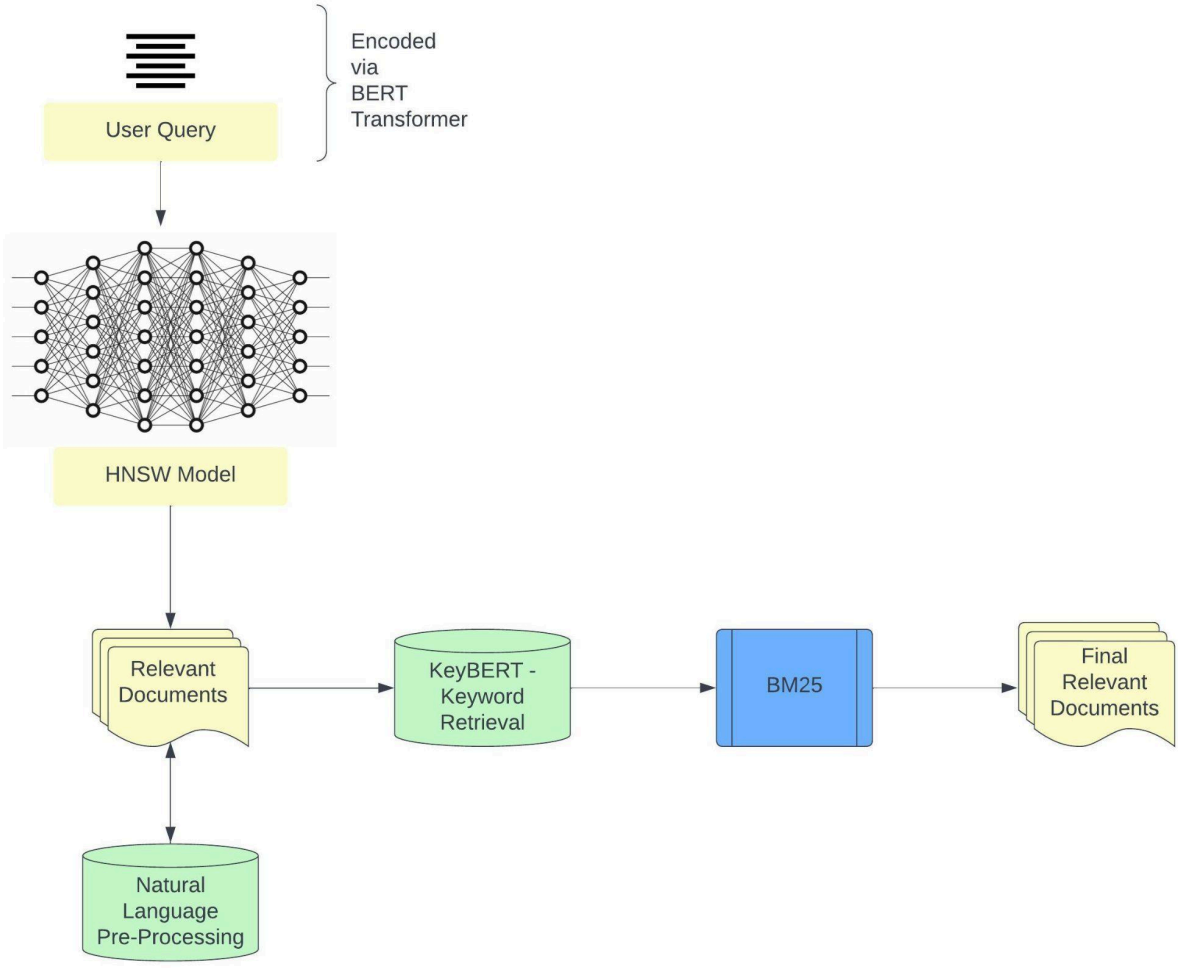
From the above diagram, the user query undergoes a transformation by the KeyBERT model.

KeyBERT is a model that likely utilizes BERT embeddings to extract keywords that are semantically relevant to the query text. This process ensures that the key concepts within the query are identified and can be used to find documents that are not just keyword matches but are also semantically relevant. The relevant documents from the BM25 process are then encoded using a MiniLM Transformer. MiniLMs is one of the fastest state-of-the-art models for natural language processing that provides deep contextual representations of text. By encoding the documents through MiniLM, the system captures the nuanced semantic relationships within the text, which goes beyond simple keyword matching. The documents are further refined through language processing. This step involves removing stopwords—words that are common and do not contribute significantly to the semantic meaning of the text—and lemmatization, which reduces words to their base or dictionary form. Once the documents have been semantically encoded and language-processed, they are input into an HNSW model. HNSW (Hierarchical Navigable Small World) is an algorithm for efficient approximate nearest neighbor search in high-dimensional spaces. The HNSW model uses the MiniLM-encoded representations of the documents to perform a similarity search, identifying documents that are semantically closest to the user query.

We observed that the Ladder model performs faster in general and better contextually than all the methods described before. This is because BM25 finds the context by identifying the documents containing the words. The HNSW finds the documents closer to the BM25 identified documents improving the search results multifold. We observed that it is better for longer queries in comparison to the Reverse Ladder method described below. We believe this can be attributed to the fact that we are identifying the common documents using text match-based approaches like BM25 which establishes the boundary for HNSW to explore comfortably. However, in shorter queries, the context is dimensionally closer and does not necessarily add any value to the ladder up.

Reverse Ladder Method

The flowchart illustrates a search engine retrieval strategy that inverts the typical order of operations seen in traditional search engines. Instead of beginning with keyword extraction and then refining the search using semantic methods, this "reverse ladder" method starts with semantic search and refines using keyword-based techniques.



The user's query is first encoded using a MiniLM transformer. This step converts the query into a vector representation that captures its semantic nuances. The encoded query is then used in an HNSW model to perform a semantic search. The HNSW model outputs a set of documents that are semantically relevant to the user's query. These documents are considered a preliminary result set, relevant based on their semantic content rather than exact keyword matching. This set of relevant documents may undergo further natural language pre-processing, which could include tasks like stopword removal, and lemmatization. This step cleans and standardizes the text, preparing it for more precise keyword analysis. Now, KeyBERT is used for keyword retrieval. It analyzes the pre-processed text of the relevant documents to extract key terms and phrases. This keyword extraction is informed by the semantic context of the documents, as KeyBERT uses BERT's embeddings to identify keywords that are representative of the main topics or themes in the text. With keywords from KeyBERT, a subsequent search or ranking is performed using the BM25 algorithm. This algorithm ranks the previously selected documents based on how well they match the extracted keywords. BM25 considers term frequency and document length to provide a score for each document's

relevance to the keywords. This final set is presented to the user as the search results.

Here, we observed that the context is established by HNSW first. It works well with shorter queries because the context is established and in the index, the similar documents stay close together. Later, the keyword extraction along with BM25 finds the intersection of the passages among all query terms. We chose 5 as a starting value for the keyword extraction. This requires more experimentation and finding the optimal value based on the query is a research area that can be explored.

Vocabulary Feedback (Pseudo - Relevance Feedback)

Vocabulary feedback in web search engine design refers to a mechanism or technique used to enhance the quality of search results and improve the user experience by incorporating user interactions and feedback into the search process. Vocabulary feedback systems generate context-aware query suggestions. When a user enters a query, the system will offer suggestions for refining or expanding the query based on similar passages. These suggestions help users better express their intent. Pseudo-relevance feedback is a technique that aims to improve the relevance of search results by analyzing the initially retrieved documents, identifying relevant terms, and using them to expand the user's query [13][14]. Our system returns a set of documents for the input query through BM25. The most important terms and relevant terms from these documents are chosen and appended to the initial query to get a modified query. This modified query serves as an input to the HNSW neural network model and returns the final set of documents which will be returned to the user. Pseudo-relevance comes into play as the system assumes that the top-k documents returned by BM25 are all relevant without the user explicitly deciding.

As an example, Let's say a user is searching for information about "sustainable energy sources" using our web search engine system.

User Query: "Sustainable energy sources"

- The system retrieves an initial set of documents using the BM25 ranking algorithm based on the original query.
- Among the initial set of documents, it identifies common terms like "wind power," "solar

energy," and "geothermal energy" as relevant terms.
- It creates a modified query by adding these relevant terms to the original query: "sustainable energy sources"

Modified Query: "Sustainable energy sources wind power solar energy geothermal energy"

The modified query is then used as input to the HNSW model and returns a final set of documents that are expected to be highly relevant to the user's query

Results

The results comprise the outputs of our pseudo-relevance feedback algorithm as well as a comparison of the output passages for the BM25 + semantic search algorithm, HNSW algorithm, BM25+HNSW ladder method, and the HNSW+BM25 reverse ladder method.

Relevance Feedback:

```
Given Query :: java
Post Pseudo Relevance Feedback Query::  java compilers programming compiler javaworld
```

```
Given Query :: java coffee
Post Pseudo Relevance Feedback Query::  coffee coffeemaker coffees brew brewing
```

```
Given Query :: java code
Post Pseudo Relevance Feedback Query::  java programming programmers programmer computing
```

```
Given Query :: java indonesia
Post Pseudo Relevance Feedback Query::  rabies indonesia jakarta indonesian papua
```

```
Given Query :: java script
Post Pseudo Relevance Feedback Query::  jscript javascript java scripts openscript
```

```
Given Query :: purple pride
Post Pseudo Relevance Feedback Query::  transgender flags lgbt pink colors
```

Queries regarding terms such as "java" encompass diverse meanings in various contexts, and a simplistic application of BM25 fails to adequately capture the nuanced nature of the input queries. Employing the approach elucidated in the pseudo-relevance feedback section, we observe how KeyBERT generates keywords based on the pertinent documents associated with queries containing such terms.

As illustrated in the aforementioned screenshots:

Query 1: "java"

The results predominantly pertain to computer science, indicative of the fact that the predominant theme across documents related to "java" predominantly involves the programming language.

Query 2: "java coffee"

The outcomes are exclusively associated with coffee and brewing, given the presence of the term "coffee" in the query. Consequently, the query becomes more focused on the interpretation of "java" in the context of coffee, aligning with the query's intent.

A comparable pattern is discernible in the remaining three queries containing the term "java," wherein the returned keywords are aligned with the contextual relevance of the term within the respective user queries.

The next segment contains a comparison of the output of the best four methods for returning the most relevant passages. There are four different types of queries we are testing it with.

1. Definition Queries: Queries whose purpose is to ask and get answers to direct questions, usually start with “what is” or “meaning of <word>”.

2. Statement Queries: These queries are direct statements that will not necessarily have a question word in them. These queries usually ask for a piece of information and are usually in the form of phrases.

3. Detailed Query: These queries are usually much longer than the average query size. A detailed query refers to a comprehensive and specific request for information or data. It involves providing thorough and precise parameters or criteria to obtain highly specific and relevant results. This type of query typically seeks in-depth information, requiring a detailed examination or analysis to fulfill the user's needs.

4. Multi-Language Queries: Multi-language queries involve the use of multiple languages in a search or information retrieval process. This type of query is designed to accommodate users who may be proficient in or prefer different languages.

Definition Queries:

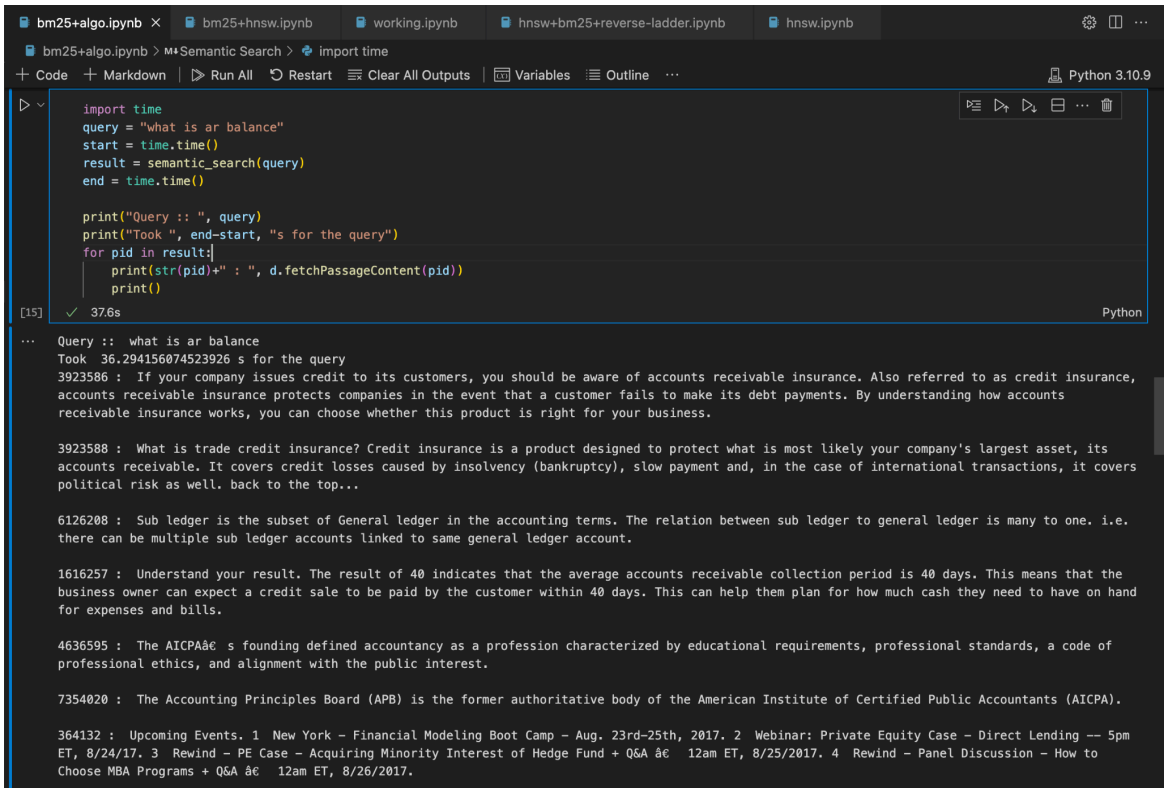
Query used: “what is ar balance”

Regarding the time required to retrieve results, a consistent pattern emerges across all queries moving forward. The swiftest passage retrieval is achieved by the BM25+HNSW method,

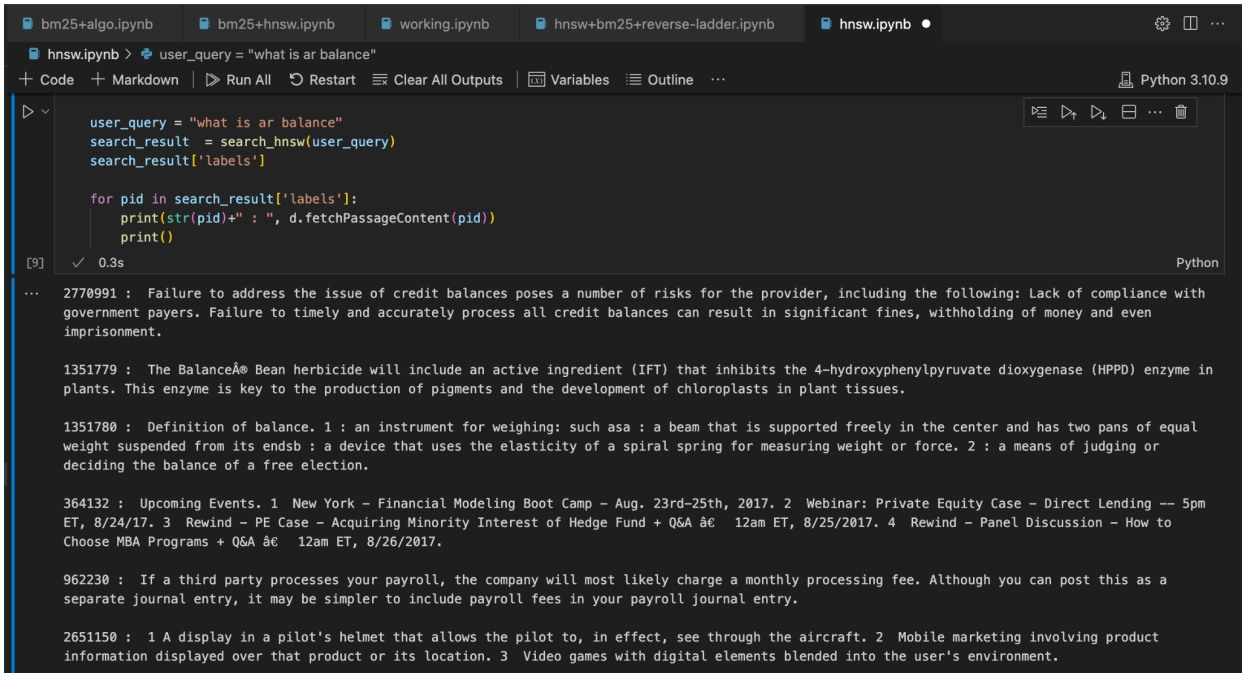
followed by HNSW in isolation, the BM25+HNSW Reverse Ladder Method, and finally the slowest performance is observed with the BM25+Semantic Search approach. It is worth noting that the reverse ladder method exhibits a marginally inferior performance compared to the standard ladder method. This discrepancy arises from the potential necessity for additional iterations of the BM25 algorithm when the initial query fails to yield any relevant passages.

In terms of the relevance of retrieved documents for a given query, all methods demonstrate commendable performance by returning documents that align well with the validation set. Moreover, it is noteworthy that the majority of the most pertinent documents are prominently positioned within the top 10 to 20 results. However, it is important to emphasize that the ordering of these results may differ among the methods, but their overall relevance remains consistently high.

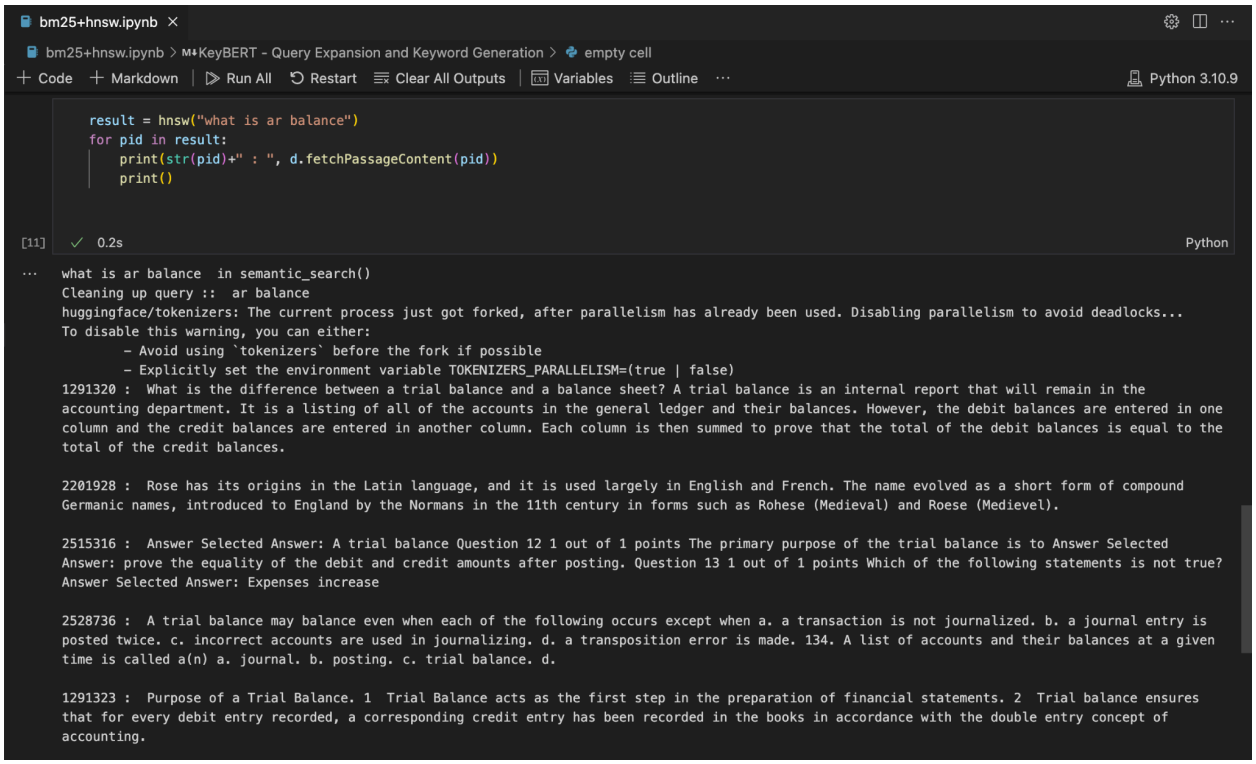
BM25 + Semantic Search:



HNSW Only:



BM25 + HNSW Ladder Method:



HNSW + BM25 Reverse Ladder Method:

```
import time
start = time.time()
query = "what is ar balance"
result = hnsw(query, 100)
end = time.time()
print("\n\n\nQuery ::", query)
print("Took", end-start, " seconds to search")
print("Results\n")
for pid in result:
    print(str(pid)+" : ", d.fetchPassageContent(pid))
    print()
```

✓ 7.3s Python

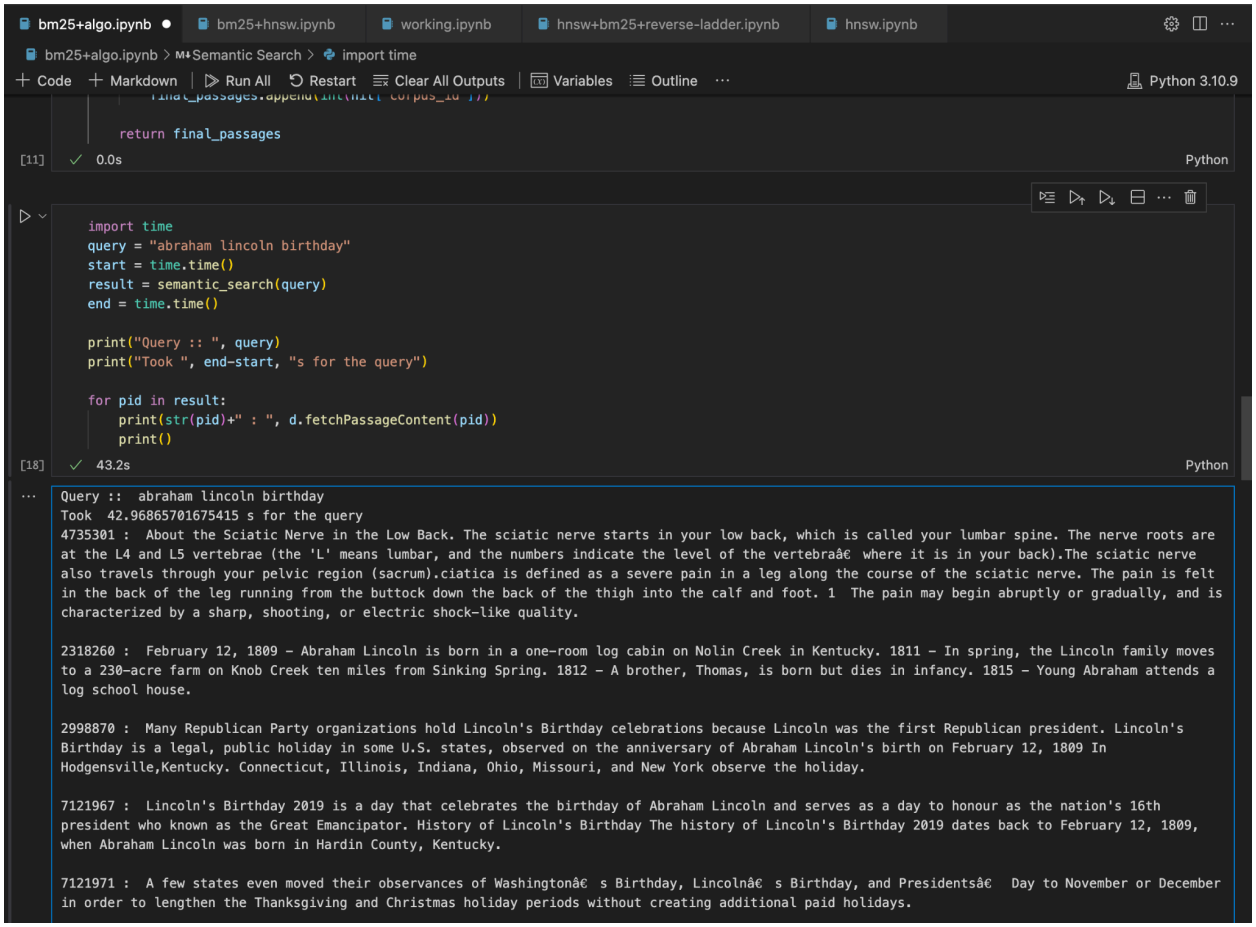
```
what is ar balance  in semantic_search()
[2770991, 1351779, 1351780, 364132, 962230, 2651150, 939873, 1678023, 939876, 939874, 1063155, 567301, 1774934, 2651144, 947878, 94005, 290746, 1702255, 960579, 939871, 538377, 2144936, 1
New Query::  accountancy balances attribute balance regulated
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
[3923586, 6105564, 5250974, 974059, 5321345, 1002002, 3660166, 820930, 820929, 3392795, 5608244, 5321343, 2373741, 364132, 6871197, 2108688, 2792812, 7278709, 2821255, 6740331, 1663742, 6



Query :: what is ar balance
Took 7.377305269241333  seconds to search
Results

3923586 :  If your company issues credit to its customers, you should be aware of accounts receivable insurance. Also referred to as credit insurance, accounts receivable insurance protec

6105564 :  The creditor may be able to charge late fees or interest if the amount is not paid by the due date. Booking a receivable is accomplished by a simple accounting transaction; how

5250974 :  Cash is an asset account with a normal _____balance. Capital is an owner's equity account with a normal _______balance. debit. When cash is paid for supplies, the supplies acco

974059 :  A transaction report is data submitted to us which contains information relating to a transaction. We use the reports to detect and investigate suspected market abuse. They may

5321345 :  b A decrease in the accounts payable and allowance accounts. c An increase in the purchase return and allowance account. d An increase in the sales return and allowance account

1002002 :  It should also be noted that this article only focuses on a few types of gases. As a more comprehensive reference, ANSI/AWS A5.32/A5.32M â€  Specification for Welding Shielding

3660166 :  As a conservator, guardian or personal representative in a supervised estate, you will be required to. file inventories and accounts with the Court. Good recordkeeping will ena
```

Statement Query

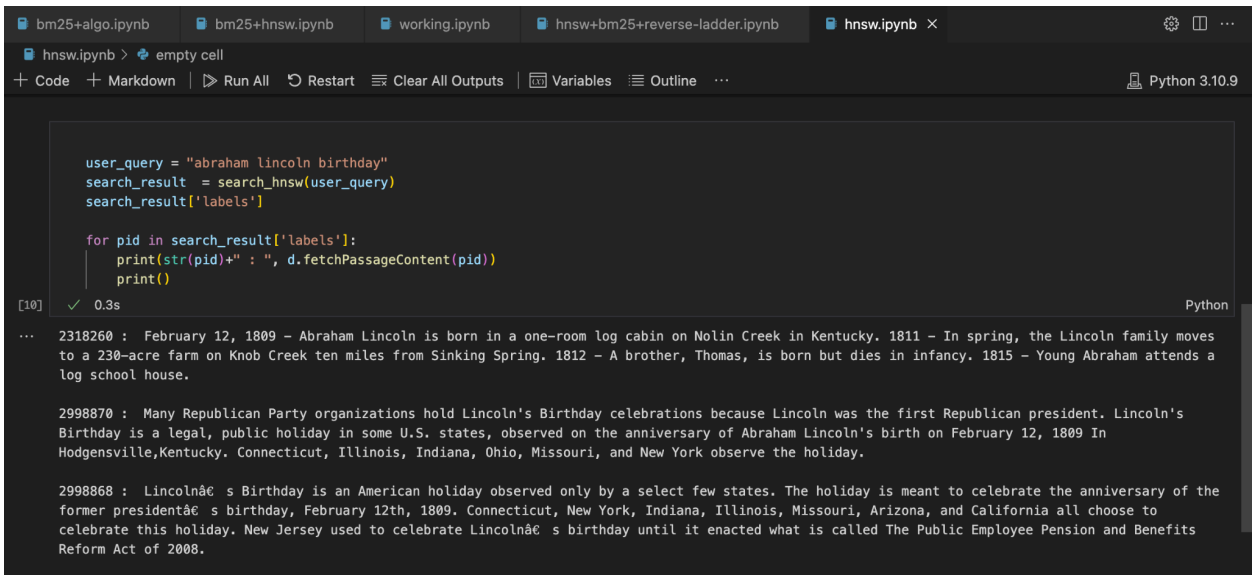
Query used: “abraham lincoln birthday”

In the context of time efficiency, a comparable pattern to the previously discussed trends persists. Both Semantic search and the BM25+HNSW Ladder method exhibit a tendency to present the foremost query result as marginally less relevant in comparison to the remaining results. Conversely, the other two methods consistently yield notably high similarity scores with the query, resulting in a collection of top passages that exhibit a commendable level of relevance.

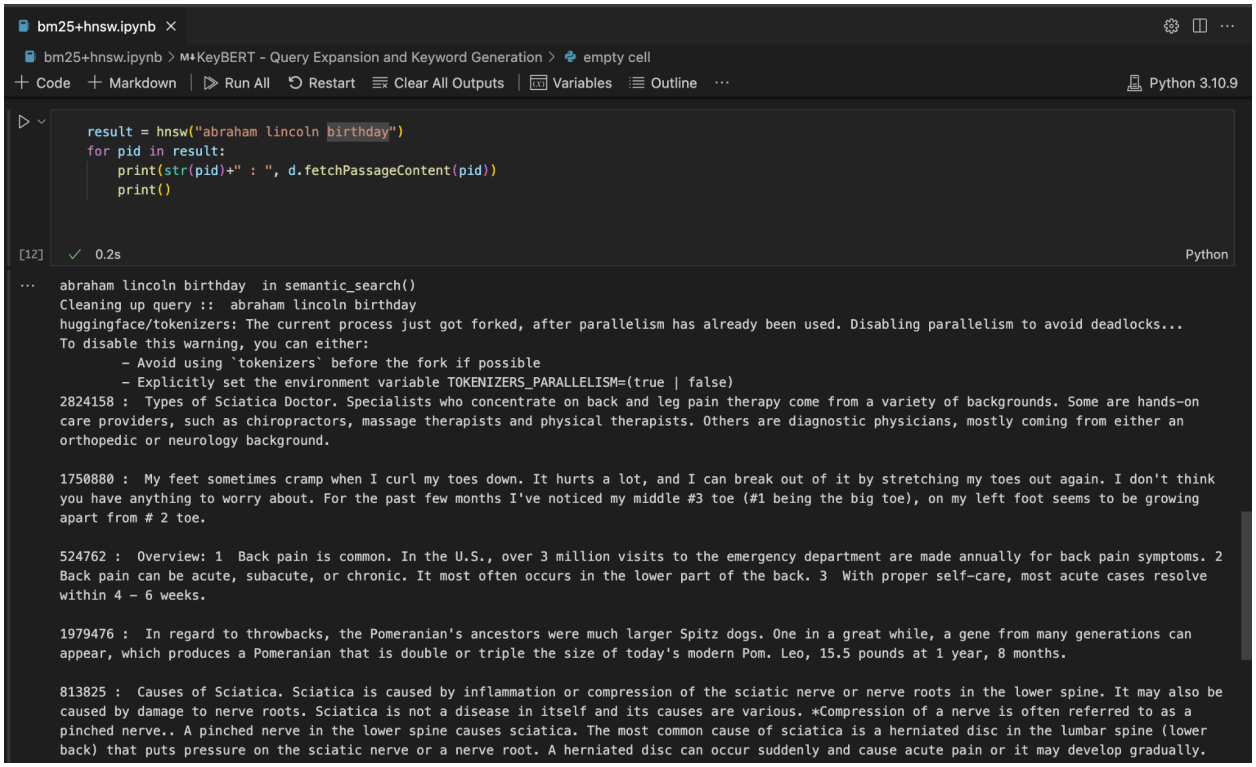
BM25 + Semantic Search



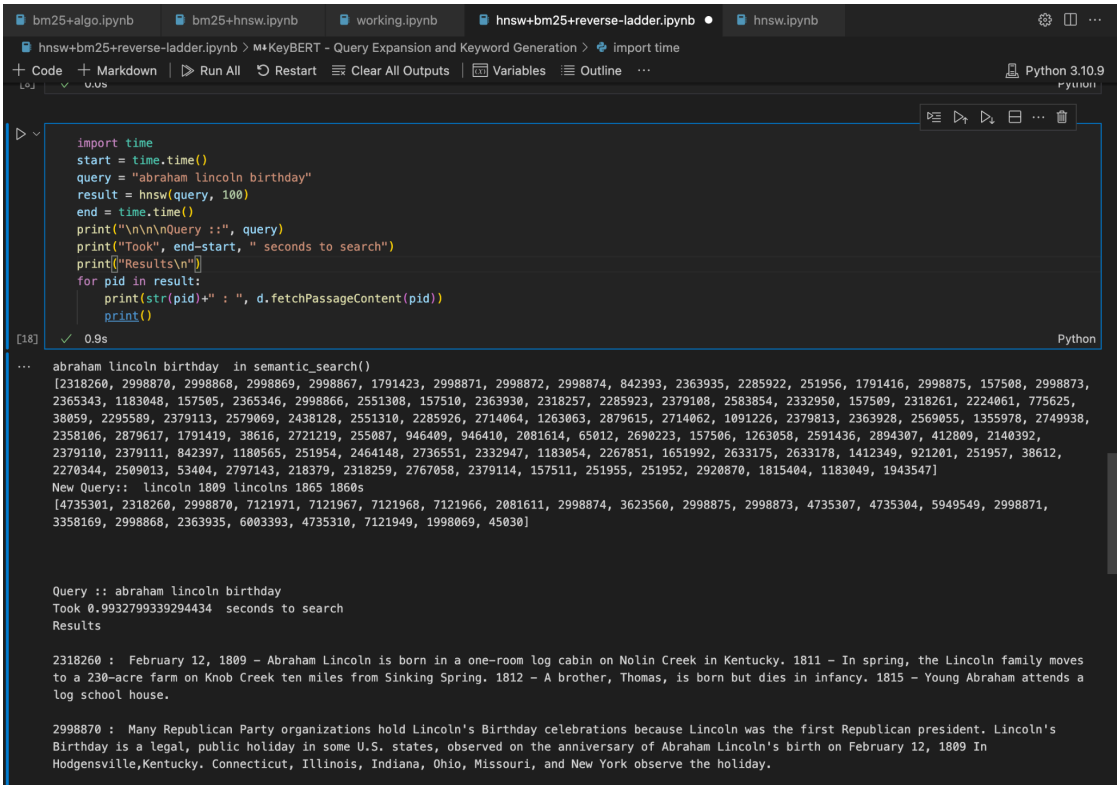
HNSW Only



BM25 + HNSW



HNSW + BM25 Reverse Ladder Method

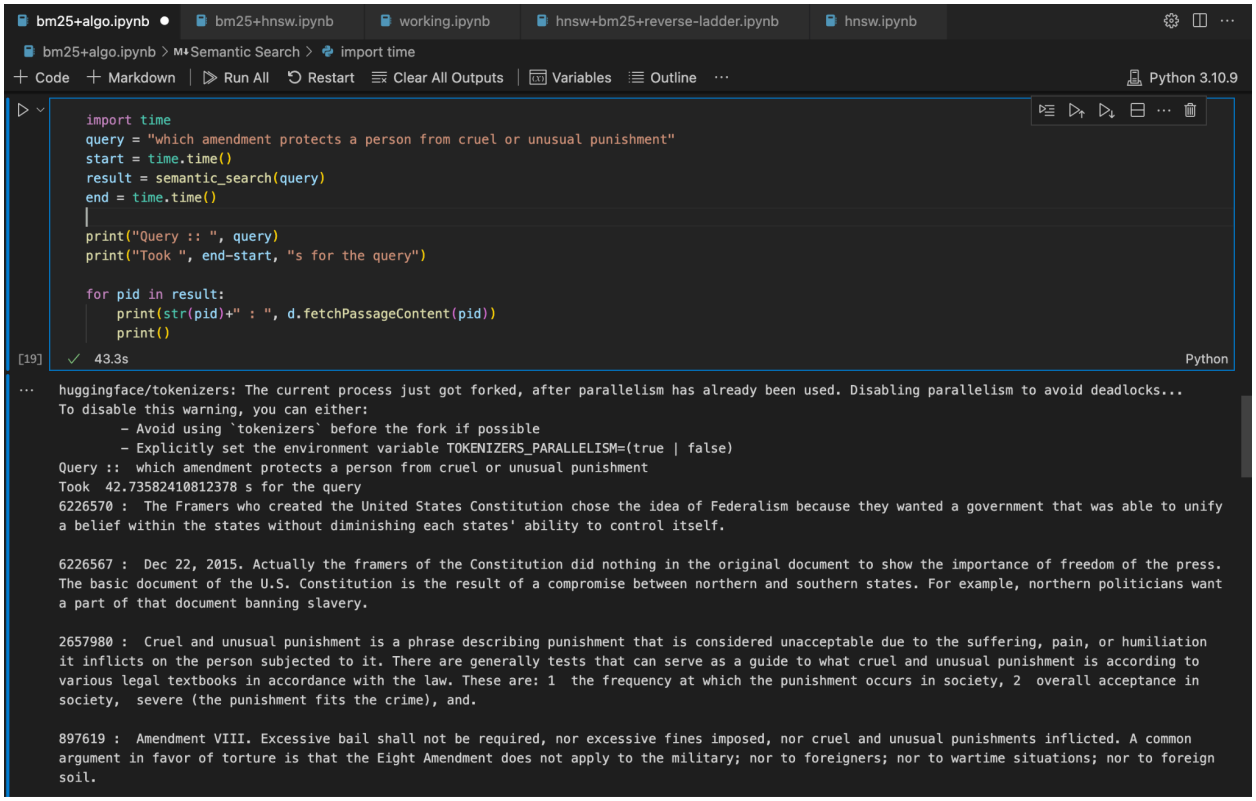


Long Queries

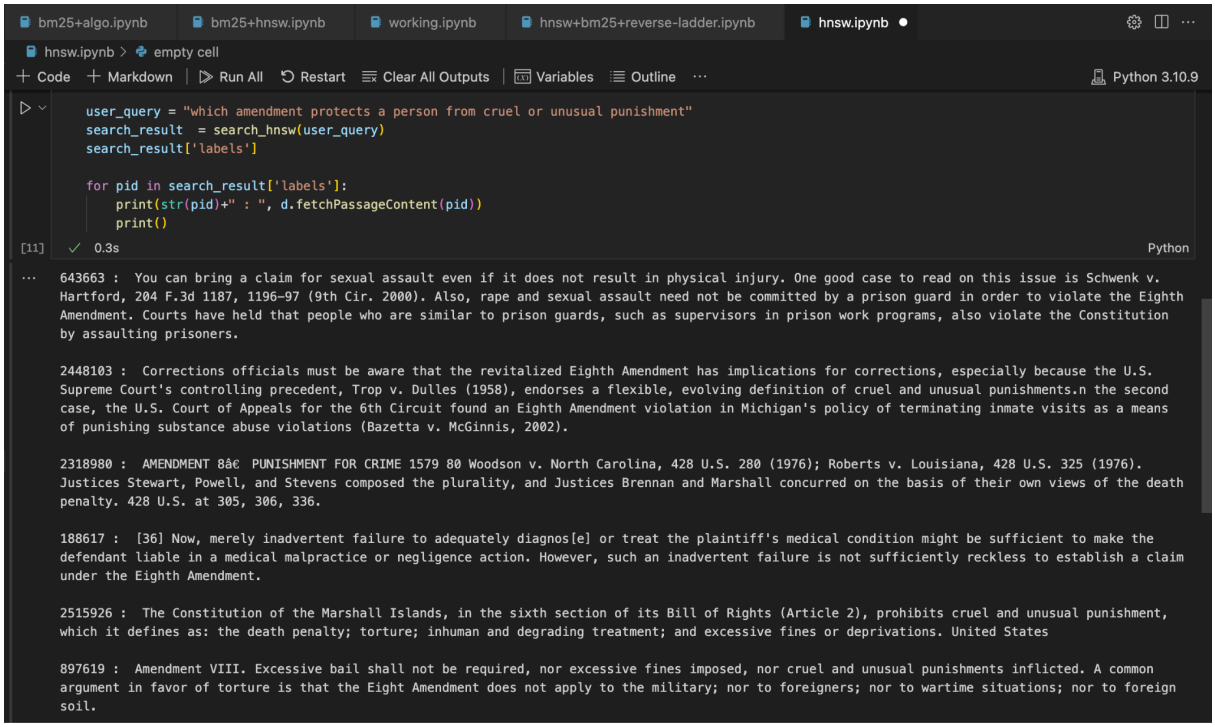
Query used: “which amendment protects a person from cruel or unusual punishment”

When it comes to longer queries, the reverse ladder method doesn’t seem to do as well as the others. This is because of the limitations of finding conjunctive passages using BM25. This is usually taken care of in the ladder method as there is another search happening after BM25, and hence it performs better. But even with this setback, the reverse ladder method does get relevant documents in its top 20 results.

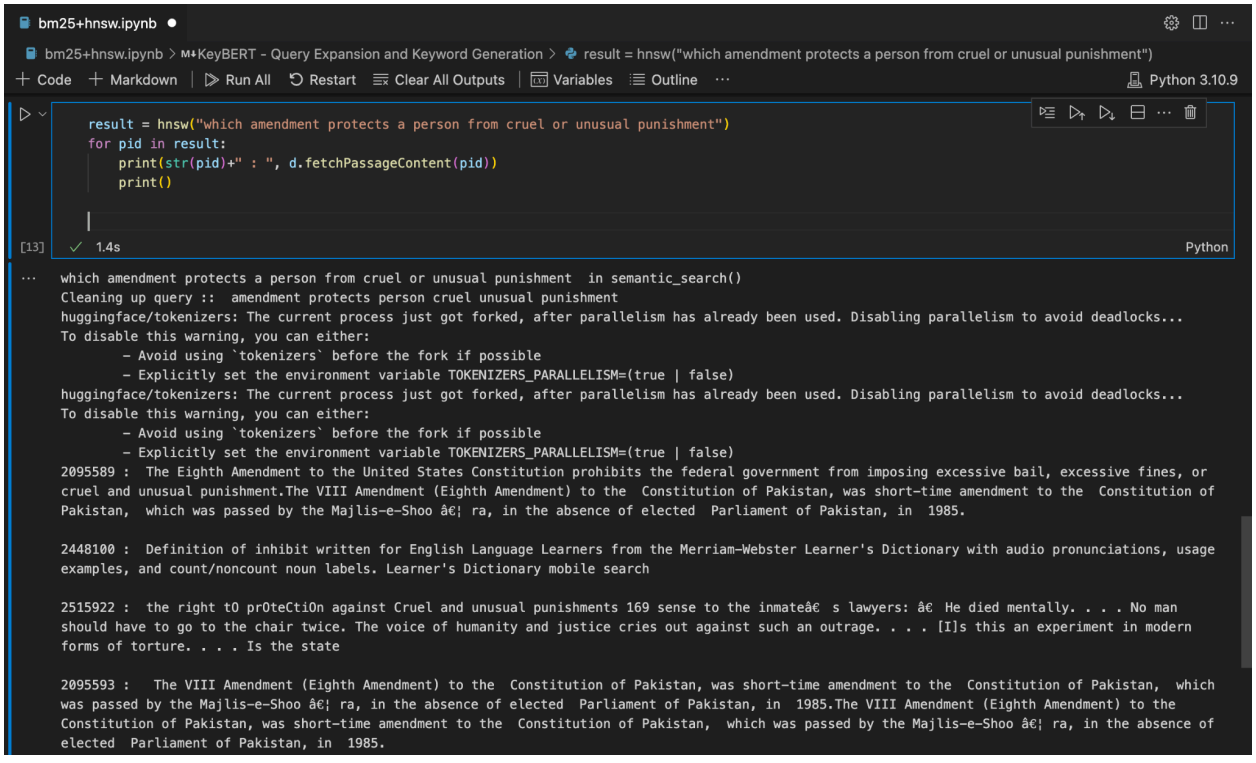
BM25 + Semantic Search



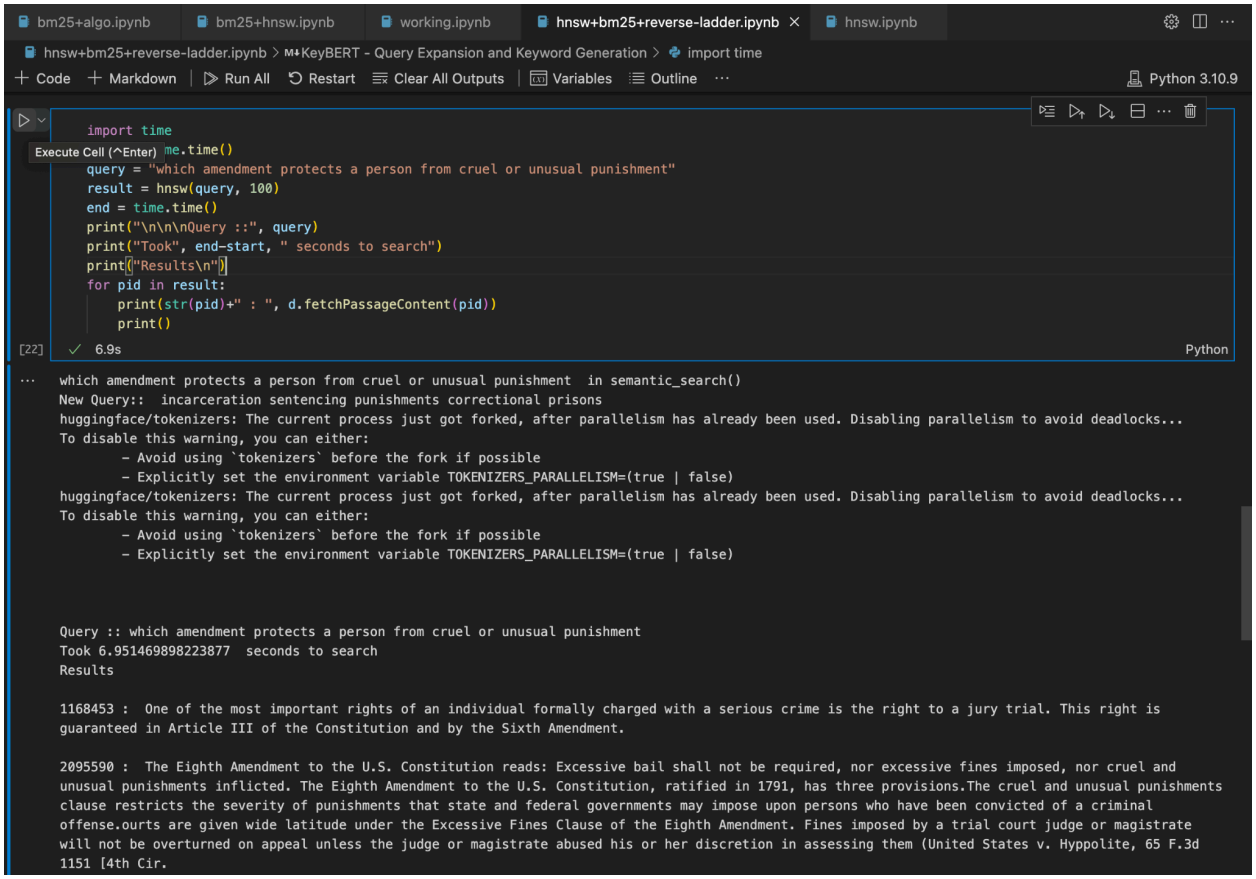
HNSW Only



BM25 + HNSW Ladder Method



HNSW + BM25 Reverse Ladder Method

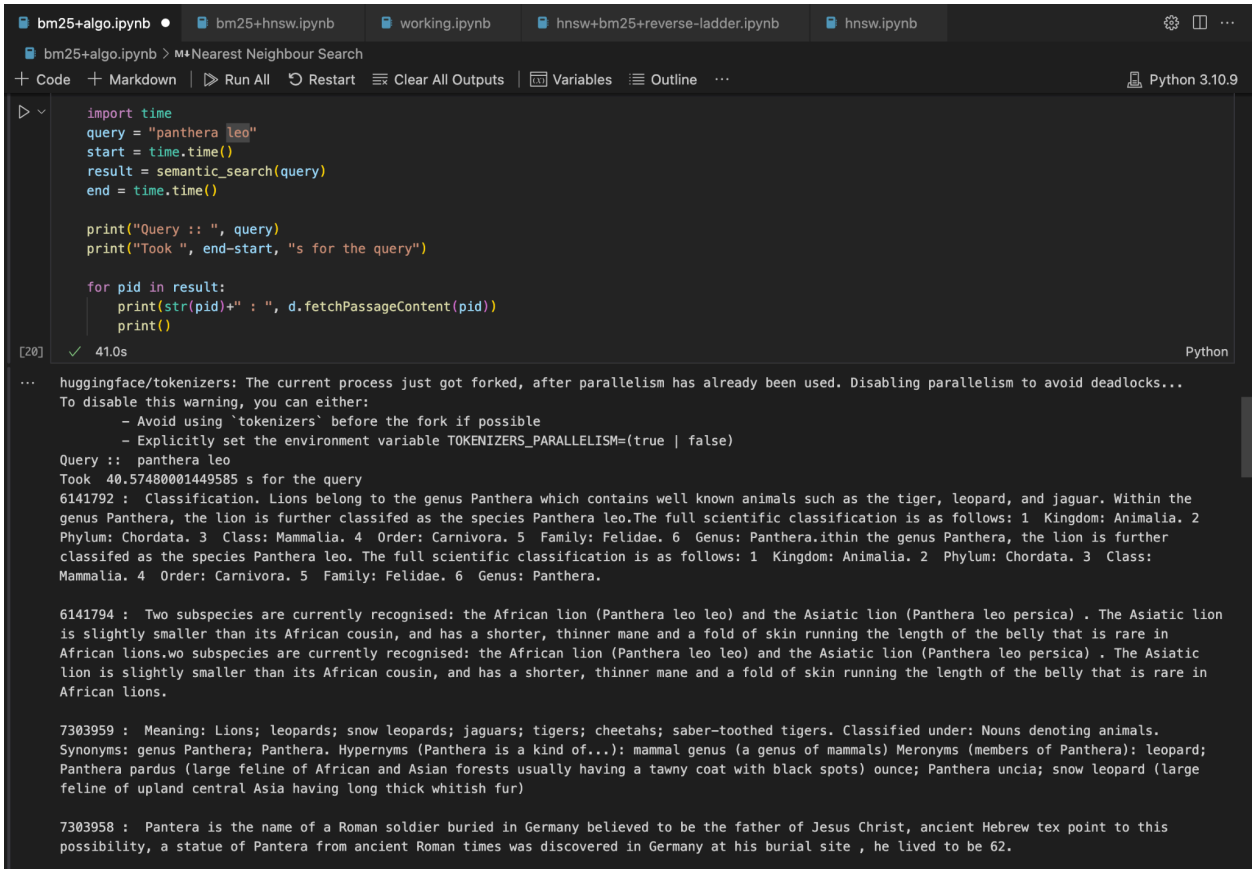


Other Languages in English

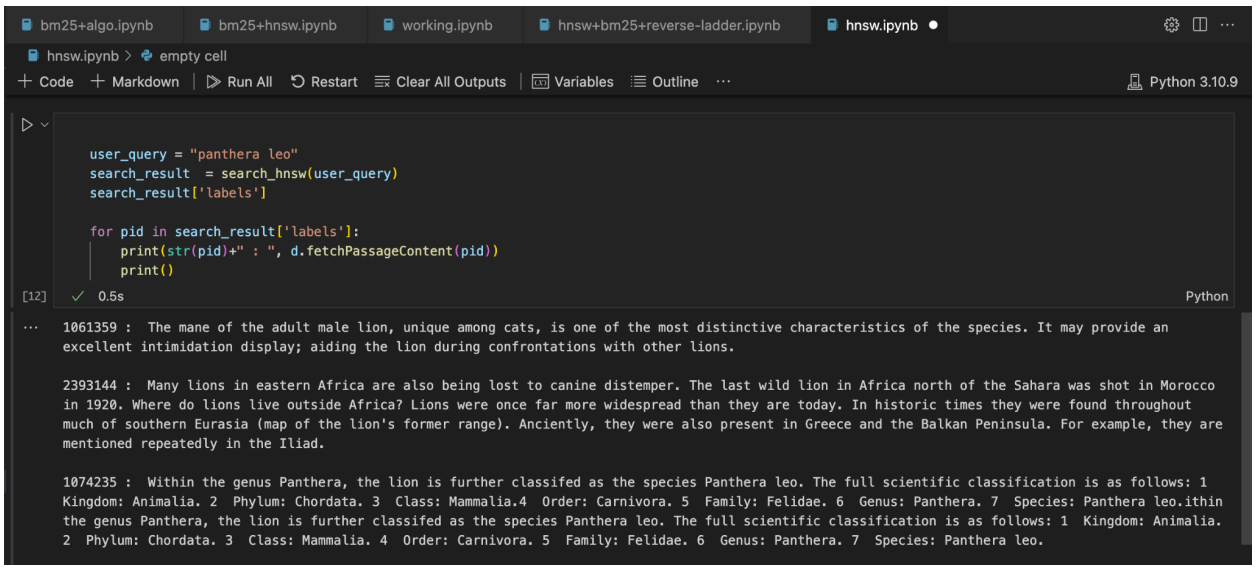
Query Employed: "panthera leo"

In this specific instance, the reverse ladder method exhibits notably superior performance when compared to the other methods. This can be attributed to a specific workflow inherent in the reverse ladder method, wherein queries of a Latin nature, such as the definition of a word, initially traverse the HNSW model. Subsequently, the HNSW model generates a set of keywords derived from the query, which are then seamlessly integrated into the query itself before being processed by the BM25 algorithm. This strategic approach enables the BM25 algorithm to effectively search for matches related to words such as "panthera" or "leo," which, in the absence of this preprocessing step, might have a heightened likelihood of being overlooked when seeking nearest neighbors rather than direct textual matches. Consequently, when dealing with queries of this nature, the reverse ladder method consistently yields optimal results.

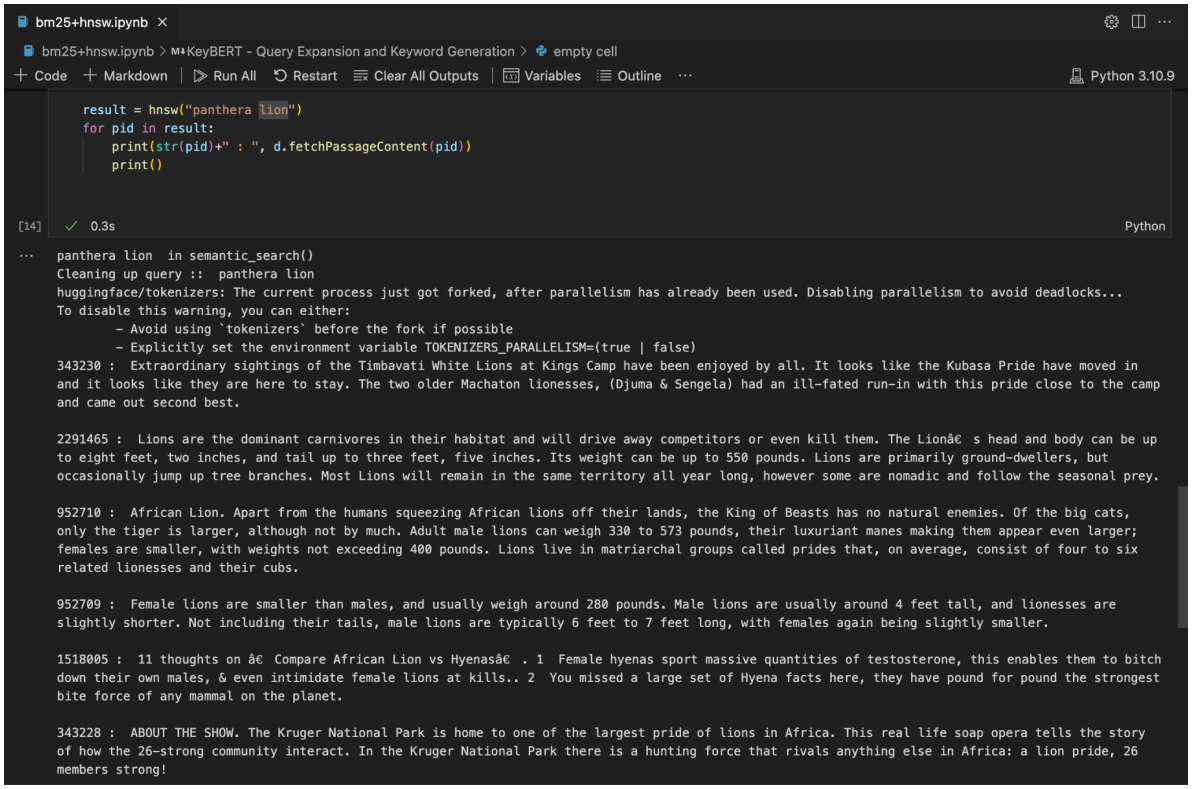
BM25 + Semantic Search



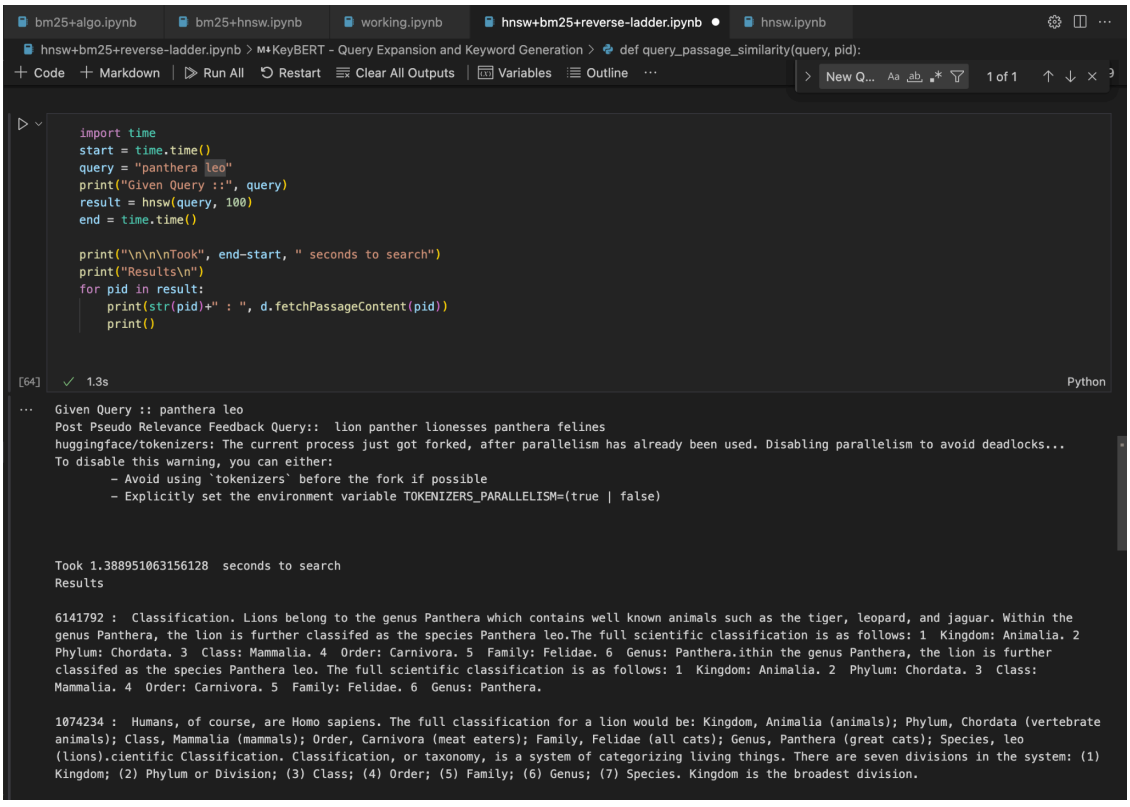
HSNW



BM25 + HNSW Ladder approach



HNSW + BM25 Reverse Ladder



We have experimented with using a similarity function (like diversity@k and cosine similarity) to check how similar the top passages are with the input query and our explanations are also partly based on that, but we found out that the method usually doesn't consider the context and returns very low scores for even the most relevant documents.

Showstoppers and Future Work

- While MiniLM is utilized in this project, future work could involve exploring and experimenting with other dense retrieval models such as BERT, RoBERTa, or even more advanced models that might be available at the time
- The project currently employs HNSW for approximate nearest neighbor search. Future work could focus on fine-tuning the parameters of HNSW or exploring other ANN methods (e.g., DiskANN, NSG) to further optimize and speed up the retrieval process.
- Further research and experimentation with evaluation metrics to ensure the effectiveness of the proposed system.
- Explore the prospect of using relevance feedback directly instead of pseudo-relevance feedback
- Investigate the adaptability of the system to multilingual or cross-lingual retrieval tasks, as this can have practical applications in diverse linguistic environments.

How to run the code

Open the .ipynb file for each of the algorithms and run the cells one at a time.

The repository structure is given below with the various files that are a part of this project.

```
- EmbeddingGeneration
   - data
       Contains the collection dataset. This is obtained from
       https://microsoft.github.io/msmarco/TREC-Deep-Learning-2020

   - embeddings
       Empty initially. Will be filled up by generateEmbeddings and embeddingConcatenator.
       It will host the individual embeddings for multiple chunks of passages.
       It will host the combined_corpus_embeddings file

   - generateEmbeddings.ipynb
       Dependencies: pip install SentenceTransformer

       Generates the embeddings in chunks

   - embedding Concatenator.py
       Dependencies: already taken care of by previous notebook

       This file combines the corpus_embeddings files from previoud file into a single file.
       If you want to skip these embedding generation phase, download the embeddings from our drive repo:
       combined_corpus_embeddings: https://drive.google.com/file/d/1wD-RjVjj7PeiENHKEeLtKMJWBI7_YtDe/view?usp=sharing
       corpus: https://drive.google.com/file/d/1ux72i05nou1BmKi2LfPebXshfm3bxIIS/view?usp=sharing
```

```
- Algorithms

    Dependencies:
        1. Make sure to unzip binIndices.zip in the same directory.
            Download this from https://drive.google.com/file/d/1swYu8i8ohV_zGEIMZ1sdMMZd_ukZSiAy/view?usp=sharing
        2. pip install nltk and related dependencies.
        3. pip install hnswlib
        4. Run g++ bm25.cpp -o bm25 -std=c++14 and generate the bm25 executable

    - maps, finalMap.bin, binIndices.zip
        Contain the inverted indices, and the dependent files for bm25

    - bm25.cpp
        Hosts the code that exposes the bm25 functionality as an executable.

    - cosineSimilaritySearch.py
        Algorithm for traditional cosine similarity search using all-MiniLM-L6-v2
        transformed passage embeddings
        Provide the query in the main function and run the file using
        python3 cosineSimilaritySearch.py

    - bm25+algo.ipynb
        Algorithm that uses BM25 followed by Semantic Search Algorithm
        Algorithm that uses BM25 followed by Nearest Neighbors Algorithm
        Query expansion logic using KeyBERT

    - bm25+algo_vocabularyfeedback.ipynb
        Algorithm that uses KeyBERT for Query Expansion and Keyword Generation
        Performs Semantic Search and Nearest Neighbour search over expanded query

    - bm25+hnsw.ipynb
        Ladder algorithm that uses BM25 to find the most relevant documents
        followed by HNSW algorithm to refine the results

    - hnswConstruction.ipynb
        Contains the logic to generate the HNSW index file for all the embeddings
        generated previously.
        Make sure to perform Embedding Generation before this step.
        If you want to skip this step, use this link to download the hnsw index bin file:
        https://drive.google.com/file/d/1zqZfbKm-tqDH7zdoHIuXX8oQiJfbFd4E/view?usp=sharing

    - hnsw+bm25+reverse-ladder.ipynb
        Inverse Ladder method that uses HNSW to get the documents first for a
        query and performs a keyword search on the returned documents to formulate
        a new query. This is passed to BM25 to get the most relevant documents

    - hnsw.ipynb
        Houses the code to test just the hnsw results for a given query

    - docContentManager.py
        Reads the corpus from EmbeddingGeneration/embeddings/corpus
        For a given pid, it returns the passage content

    - diversityScore.py
        Hosts the importable functions to calculate the diversity between
        passages returned by the various algorithms and the similarity score
        between a passage and the query.
```

References

1. Bajaj et al. (2016) in "Ms marco: A human generated machine reading comprehension dataset," published in arXiv preprint arXiv:1611.09268.
2. Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
3. Reimers and Gurevych (2019) introduced "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" in the Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, published by the Association for Computational Linguistics. https://arxiv.org/abs/1908.10084
4. Wang, Wenhui, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers." *Advances in Neural Information Processing Systems* 33 (2020): 5776-5788.
5. Kassim, Junaidah Mohamed, and Mahathir Rahmany. "Introduction to semantic search engine." In *2009 International Conference on Electrical Engineering and Informatics*, vol. 2, pp. 380-386. IEEE, 2009.
6. Liu, Jiawen, Zhen Xie, Dimitrios Nikolopoulos, and Dong Li. "{RIANN}: Real-time incremental learning with approximate nearest neighbor on mobile devices." In *2020 USENIX Conference on Operational Machine Learning (OpML 20)*. 2020.
7. Han, Jianfeng, Xuefei Guo, Runcheng Jiao, Yun Nan, Honglei Yang, Xuan Ni, Danning Zhao et al. "An Automatic Method for Delimiting Deformation Area in InSAR Based on HNSW-DBSCAN Clustering Algorithm." *Remote Sensing* 15, no. 17 (2023): 4287.
8. Issa, Bayan, Muhammed Basheer Jasser, Hui Na Chua, and Muzaffar Hamzah. "A Comparative Study on Embedding Models for Keyword Extraction Using KeyBERT Method." In *2023 IEEE 13th International Conference on System Engineering and Technology (ICSET)*, pp. 40-45. IEEE, 2023.
9. Sammet, Jill, and Ralf Krestel. "Domain-Specific Keyword Extraction using BERT." In *Proceedings of the 4th Conference on Language, Data and Knowledge*, pp. 659-665. 2023.
10. KeyBERT Github repository: https://github.com/MaartenGr/KeyBERT
11. Broder, Andrei Z., Peter Ciccolo, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel. "Search advertising using web relevance feedback." In *Proceedings of the 17th ACM conference on information and knowledge management*, pp. 1013-1022. 2008.

12. Ruthven, Ian, and Mounia Lalmas. "A survey on the use of relevance feedback for information access systems." *The Knowledge Engineering Review* 18, no. 2 (2003): 95-145.
13. Xu, Yang, Gareth JF Jones, and Bin Wang. "Query dependent pseudo-relevance feedback based on wikipedia." In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 59-66. 2009.
14. Reuben, Maor, Aviad Elyashar, and Rami Puzis. "Iterative query selection for opaque search engines with pseudo relevance feedback." *Expert Systems with Applications* 201 (2022): 117027.
15. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press