# Information Retrieval project

Our project aims to construct a search engine encompassing the entire English Wikipedia corpus, which consists of over 6 million documents. The objective is to develop a search engine that takes queries as input and efficiently outputs the most relevant documents in the shortest possible time.

Throughout the project, we experimented with various assumptions based on findings to build the search engine in the fastest and most accurate way possible.

Each Wikipedia document is characterized by three components: title, body, and anchor text. We create an inverted index for the two primary components, title and body. Additionally, using the anchor text, we created a dictionary that contains rankings for each document based on page rank.

Ultimately, the search function returns (id, title) for the most relevant documents. After identifying the top 50 relevant documents, we need to map them to id, title pairs. Therefore, we stored a dictionary for each document with id, title as a pkl file in GCP bucket.

Page Rank Integration: To enhance the relevance of retrieved documents, we prioritize those with high page rank ratings. This integration not only emphasizes relevance but also considers the authority by the internet link network, ultimately improving search result quality. The addition of page rank ratings significantly improved result accuracy. To use page rank into document ranking optimally, we experimented with assigning varied weights to the page rank rating. We observed that overly high page rank values led to the retrieval of documents with high ratings but reduced relevance to the query. Conversely, assigning too low a weight resulted in a decline in the quality of retrieved documents. Ultimately, we found that the most effective weighting method involves taking the base-10 logarithm of the page rank and applying it to each document's rank.

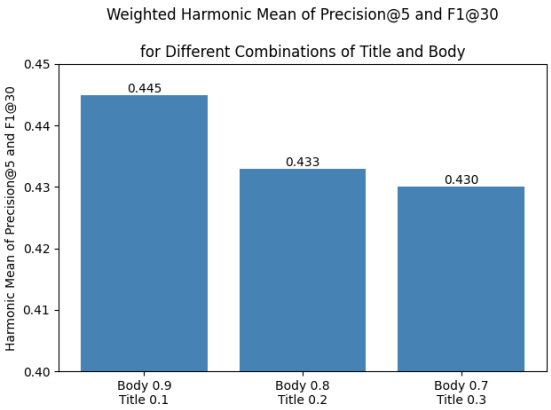
Page Views Consideration: After computing page views for all documents, we decided not to use them in our system. Despite multiple experiments, we found that page views did not notably enhance result quality. Our observations revealed that relying on page views did not consistently improve relevance rankings, as popular pages did not consistently contain the most relevant information.

Similarity Method: In our quest for accurate document ranking, we explored similarity methods such as cosine similarity and BM25. BM25 emerged as the preferred method due to its ability to weigh term frequencies, normalize document length, and consider relevance to the entire corpus. To calculate the BM25 score, we utilized the following formula: $score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot(1-b+b\cdot\frac{|D|}{avgdl}}$

To optimize search performance, we conducted precomputations, including a part of calculations of the BM25 that independent of specific queries, during the creation of the inverted index. In each inverted index, we stored the following fields:

1. df - document frequency per term.
2. Term total - total frequency per term.
3. Posting list - stored posting list per term during index building (internally) as it would otherwise be too large to store in memory.
4. Posting locs - mapped a term to posting file locations, a list of (file name, offset) pairs.
5. N - the number of documents in the corpus.
6. Similarity - $k \cdot (1 - b + b \cdot \frac{|D|}{avgdl})$ for each document.

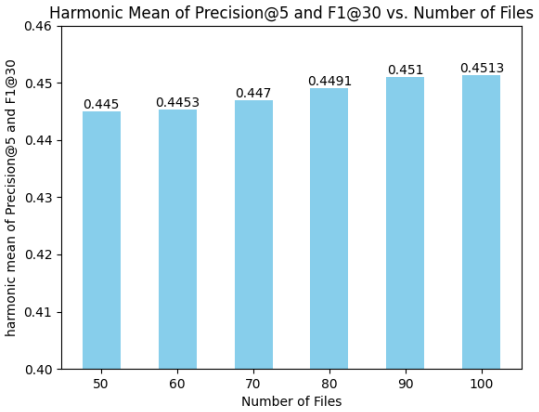
Optimizing BM25 Scoring: To identify the best weight distribution for scoring BM25 in the body and title, enhancing the retrieval of the most relevant documents, we conducted multiple iterations with varying weight combinations such as Title 0.1, Body 0.9; Title 0.2, Body 0.8; and Title 0.3, Body 0.7. The experiment indicated that the first combination yielded the optimal result in the harmonic mean of Precision@5 and F1@30 (0.445), surpassing the lower results of the other combinations (0.433, 0.430). Consequently, we assigned a weight of 0.1 to the title and 0.9 to the body.

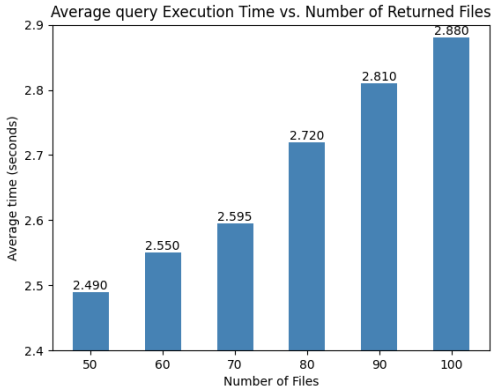



Query Length Impact: In our pursuit of achieving the utmost accuracy, we conducted experiments to determine suitable weights for the body and title based on query length. Our findings revealed that when the query comprises

less than 3 words, allocating a higher percentage to the title (0.9 compared to 0.1 for the body) yielded greater accuracy. This is attributed to the increased significance of each word in shorter queries. Conversely, for longer queries (3 words or more), assigning higher percentages to the body (0.9 compared to 0.1 for the title) resulted in improved accuracy.

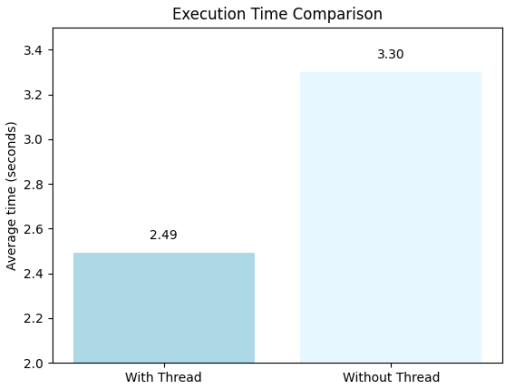
Document Quantity and Search Efficiency: Additionally, we assessed how the quantity of returned documents impacts accuracy and processing time in our search function, designed for up to 100 relevant documents. Limiting returns to 50, 60, 70, 80, 90, and 100 documents, we observed that more returns led to a higher harmonic mean of Precision@5 and F1@30, logically providing a broader pool of potentially relevant documents. The algorithm's accuracy improved with a more extensive array of results. Regarding average query time, it increased with more returns, likely due to additional sorting and mapping processes. We found a correlation between the number of returned documents and average processing time. Returning 50 documents yielded the shortest average time (2.490 seconds), balancing a favorable harmonic mean with a reasonable average query time.



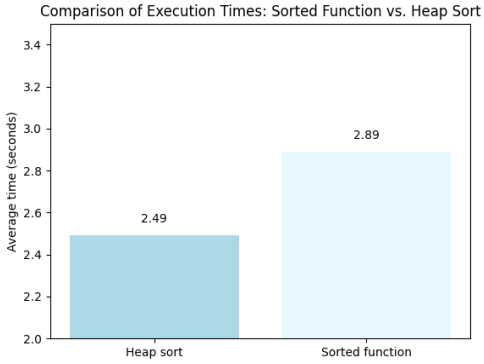

Multithreading for Parallel Execution: To further streamline search time, we explored the use of multithreading for parallelizing actions. The computation of the BM25 score and page rank for the title and body was conducted in

parallel threads. Multithreading significantly reduced the average query time, prompting the creation of separate threads for the body and title, which were merged upon completion. Attempts to partition the posting list into batches did not yield significant improvements in search time.




sorting:

We evaluated the impact of different sorting methods on time efficiency by comparing the average time when using Python's built-in sorting function versus heap sort. It was observed that heap sort significantly reduces the required time, prompting us to choose heap sort for sorting the results.




Success and Insights from a Well-Performing Query : “When was the Gutenberg printing press invented?”

harmonic mean of Precision@5 and F1@30 : 0.712

This query yielded excellent results due to its unique combination of words found in a limited number of documents, increasing the likelihood of returning highly relevant results. Our effective similarity method and overall scoring calculation demonstrated outstanding performance in this case.


Deconstructing a Low-Performing Query:

Query: “Who is considered the "Father of the United States”

harmonic mean of Precision@5 and F1@30 : 0

This query performed poorly due to the common occurrence of the words "Father" and "United States" in a large number of documents (approximately 2 million for "Father"). The abundance of results containing these words did not necessarily relate to the intended meaning of the query. Unfortunately, our system, lacking models considering concepts like LSI or indices for phrase combinations, missed the query's semantic nuances and failed to retrieve relevant documents.