Web Search and Information Retrieval in the Internet

Mateusz Janowski
Instituto Superior de Engenharia do Porto, Master’s Degree in Computer Engineering, Data Engineering,

Porto, Portugal
1240288@isep.ipp.pt

Louis Scherpereel
Instituto Superior de Engenharia do Porto, Master’s Degree in Computer Engineering, Data Engineering,

Porto, Portugal
1242390@isep.ipp.pt

Luka Ćavar
Instituto Superior de Engenharia do Porto, Master’s Degree in Computer Engineering, Data Engineering,

Porto, Portugal
1242332@isep.ipp.pt

Filip Orlikowski
Instituto Superior de Engenharia do Porto, Master’s Degree in Computer Engineering, Data Engineering,

Porto, Portugal
1242389@isep.ipp.pt

I. Introduction

Today’s digital world provides so much information on so many subject that locating something relevant is both a problem and a need. Every second, the number of searches executed by search engines is equal to billions, guiding the users into the huge ocean of data the Internet possesses.

But how is this possible behind the scenes? Which mechanism is implied for billions of web pages and provides the most relevant ones in milliseconds? This study sheds the light on some sophisticated aspects of Information Retrieval (IR) systems on the Internet, focusing on the algorithms, indexing and ranking techniques driving contemporary search engines [1][6].

The study describes crawling, indexing, relevance scoring and personalization. It also explains the core features of web searching, that contribute to its efficiency and effectiveness.

Understanding of these concepts will not only reveal the cleverness of search engines, but also elucidate the way information is structured, retrieved, and presented in today's connected digital landscape.

II. Search Engines

Search engines are the large-scale information retrieval systems. They process and index web-based information to provide specific answers to the users’ queries[2][6]. Technically, they use a blend of distributed computing, algorithmic logic and data structures for processing and analyzing enormous amounts of online information [2]. In order to determine the user intent, the search engine processes the input of retrieved query through operations like entity recognition, tokenization, and stemming. Next it fetches relevant documents from its pre-processed index and ranks the responses through a set of ranking algorithms that evaluate numerous signals of relevance. Such systems, typically involve inverted indexes, load balancing, and real-time query processing, and also are fault-tolerant, scalable, and fast.

III. Crawling

A. *What is Web Crawling?*

The easiest way to explain the term “web crawling” is to say that it is the task of finding and indexing web pages [3]. However, what actually stands behind this term is much more.

Crawlers - also known as bots or spiders - are specially designed computer programs used for the web scanning process [4][5]. They start by visiting pages from a predefined list of known URLs. These pages are often referred to as seeds, because they serve as the starting point from which the entire network structure is gradually discovered.

One of the main goals of crawling is to build and continuously update the structure of the web. How does this happen? While crawling seed pages, the crawler identifies hyperlinks to other pages embedded within them. By following these links recursively, it constructs a map of interconnected pages — showing how one page leads to another [3].

The aspect that is most important in Information Retrieval systems, and is also handled by crawlers, is the analysis of web content. Crawlers not only discover links to other websites but also extract the actual information contained on each page and forward it to the indexer, which prepares it for later searching and ranking [4].

B. *How Content is fetched from a web page?*

The crawler begins with a seed URL that is known. After retrieving the page's content, it parses the HTML to extract the content itself as well as any links to other pages [3].

To collect data from the web, a crawler must be able to locate and download online resources. This involves several technical processes that allow it to interact with websites similarly to a regular browser, but automatically and on much larger scale [4][5].

Let's focus on the steps taken to fetch the web page content.

Step 1: DNS Resolution
Firstly the crawler sees only the URL. We will work on this example:

```
http://www.example.com/Resources/data.json
```

In order to "speak" with the server it first requests for the www.example.com IP address, as computers communicate using IPs, not names.

This process is called DNS resolution and has two possible scenarios:

- Check local cache. If the system already knows the IP address from the request it returns the desirable IP.

- If IP address is not cached, it sends a request to DNS resolver, which asks who handles a root server (.com), a TLD server (where is example.com) and an authoritative server (what's the IP of www.example.com)

When the resolvers gets the IP address:

```
www.example.com → 93.184.216.34
```

it sends it back to the crawler, and both sides can store (cache) the results for future use.

Step 2: Opening the connection
When the crawler knows the IP address, it opens a TCP connection to it, with port 80 - standard for HTTP requests.

Step 3: Sending the HTTP Request
To retrieve the desired resource the crawler usually uses the GET method, which is designed to request data from a specified resource without modifying it.

The HTTP request looks like this:

```
GET /Resources/data.json HTTP/1.1

Host: www.example.com

User-Agent:     Mozilla/5.0     (compatible;
MyCrawler/1.0)
```

First line specifies the used method and the path to the resource on the web server. It also denotes the version of the HTTP protocol used.

Second line specifies the domain name of the server being requested. This header is mandatory in HTTP/1.1.

Finally, the third line identifies the client software initiating the request.

Step 4: Receiving the response
The server processes the request and sends back a response:

```
HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: 1256

Last-Modified: Tue, 25 Mar 2025 10:10:00 GMT
```

Then comes the actual content of the file, that was supposed to be obtained. The crawler send this data to the indexer.

*C. **URL Frontier***

When crawling a web page, all hyperlinks encountered on the page are "harvested" and placed in a queue of URLs to be visited, called the URL frontier [3].

The URL frontier is a priority-managed queue storing URLs that were discovered while parsing but have not been fetched yet. This structuring allows the crawler to handle and decide the manner in which it accesses the pages, thus performing a controlled and structured search over the web.

For every new page that the crawler downloads:

1) Extract new URLs from its content,
2) Apply filters (to get rid of duplicates or disallowed URLs, for instance),
3) Add valid links back into the frontier.

This forms a recursive loop of exploration, where each new page could also lead to other pages, and the crawl graph will grow over time.

URL Filtering, duplicate handling

Before adding URLs to the frontier, the crawler applies filtering mechanisms to eliminate irrelevant, disallowed, or redundant links. This includes removing duplicate URLs (e.g., with different tracking parameters) and normalizing them into a canonical form. Such filtering reduces unnecessary requests and ensures efficient resource usage.

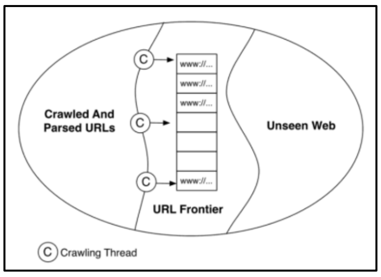


Figure I-1: The crawling process, source: [2]

To promote effective and fair behavior in the frontier, more policies arise, e.g.:

1) Relevance, freshness, or importance (one at a time)
2) Politeness constraints
3) Re-evaluate scheduling if the content is updated often.

D. *Crawler directives*

To ensure ethical and responsible indexing, web crawlers must follow the guidelines provided by site owners. These guidelines are called crawler directives and there are several common ways to implement them:

a) robots.txt file

Websites use a file called robots.txt that is located at their root (e.g., www.example.com/robots.txt), to communicate with web crawlers. This file specifies which parts of the site crawlers are allowed or disallowed to explore [1][11].

```
User-agent: *

Disallow: /private/

Allow: /public/
```

- In the first line, * indicates that all crawlers have to follow this rule.

- The next line instructs bots not to visit URLs that are located under *private*.

- The last line instruct bots which URLs are allowed to be crawled, in this case only the ones located under *public*

Before retrieving any pages, crawlers read this file and modify their behavior accordingly. It is crucial to remember that robots.txt is only recommended and not legally binding; compliance by the crawler is up to them [1].

In the *robots.txt* file, website owners can also include a reference to an XML sitemap, which helps crawler analyze URLs more easily. That is especially useful if the site has dynamic or deeply nested pages. An example of such reference looks like this:

```
Sitemap: https://www.example.com/sitemap.xml
```

b) *meta tags in HTML*

In addition to robots.txt, webmasters can use HTML <meta> tags to embed crawler directives straightly within a web page:

```
<meta name="robots" content="noindex, nofollow">
```

This tag informs crawlers:

- *noindex*: This page should not be indexed by search engines.

- *nofollow*: Avoid clicking on any of the page's links.

These tags provide more precise control over crawling and indexing actions specific to a page [2][11].

c) *HTTP headers*

Another way to guide the crawler is by using HTTP response headers:

```
X-Robots-Tag: noindex, nofollow
```

The main advantage of this kind of technique is that it permits directives for non-HTML resources. Although it is set on the server side, this functions similarly to the HTML meta tag [2][11].

IV. INDEXING

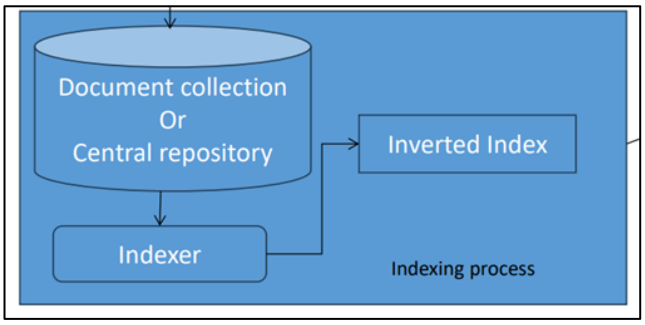


Figure II-1: The indexing process [24]

Indexing is an important process in an information retrieval system that enables effective searching and retrieving of relevant documents from a vast collection. Users of information desire accurate and speedy results, which would be virtually impossible without a systematically organized indexing system. The diagram below shows the process of indexing, involving data arrangement and organization to optimize search effectiveness. The most important elements of this process are a collection of documents or a database, an indexing mechanism, and an inverted index, all of which are essential to facilitate easy information retrieval.

A. Document Collection: The Foundation of Indexing

The document set or central repository is the basis of the retrieval system [7]. It holds a vast collection of documents that are to be indexed and processed for easy searching. The documents may be in many different formats such as text files, PDFs, web pages, and other electronic documents. The central repository is a bit like a storage warehouse where the information is collected prior to processing. Were it not for this collection, there would be nothing to search for information, so it is the crucial first part of the indexing process. Since the volume of digital information is continually growing, it is even more necessary to have an organized repository.

After the documents are put into storage, the responsibility falls on the indexer to work on them. The indexer reads through the content of each document, recognizes significant terms, and pulls out significant information that will be used to index them. In order to make the pulled-out information more effective, the indexer applies some techniques that include tokenization, stemming, and stop-word removal [8]. Tokenization splits up the text into separate words or phrases so that it is simple to map out words of interest. Stemming brings words down to the root of the word so that words with the identical meaning, i.e., "running" and "run," are handled as the same. Stop-word removal removes common words like "the" and "is" that do not hold a lot of value to search relevance. The processes make the index compact yet filled with the best of every document.

B. The Inverted Index

Once the indexer has retrieved and processed the relevant terms, it puts them into an inverted index. An inverted index is a key component of a good information retrieval system because it allows for fast lookups and quick access to relevant documents [8]. An inverted index does not function in the same way as regular indexes that put documents into a linear order; an inverted index puts data based on keywords, mapping each term to the documents in which it appears. This organization highly improves search efficiency by the power of the system to retrieve results without having to scan the entire repository. Instead of searching thousands or even millions of documents, the search engine directly accesses the ones that contain the words being searched [10]. This process saves much search time as well as computational power and is thus significant in handling large-scale data.

The importance of the indexing process cannot be over-dramatized because it is the pillar of any modern database or search engine system. Searching for some information in the event of defective indexing would be wastage of time, irritating, and inefficient [7]. Try to visualize a user looking for a particular article in millions of documents. Without the inverted index, the search system would need to read each document separately, which would be hours or days. But with the inverted index, the system can find the matching documents in the blink of an eye, read them back in milliseconds, and deliver them to the user in a neatly assembled package [9]. That is the magic that makes Google, library catalogs, and company information systems so powerful yet convenient.

C. Indexing in Large-Scale Digital Environments

Indexing becomes particularly crucial in big virtual spaces, i.e., business systems, web search engines, and university databases. Companies dealing with huge users' information, researches, or web sites must depend on sophisticated indexing to be able to maintain their info on a par level. Without indexing, the systems would freeze with latencies and faulty searches. Businesses, researchers, as well as ordinary users, can benefit from the productivity of indexing in the network era. Furthermore, with information continuing to grow at an exponential rate, innovations in indexing methods are on the horizon to cater to the immense number of fast and accurate information seeking requests [10].

Another key aspect of indexing is its influence on ranking and relevance. Indexed documents are not all the same; search engines rank results in terms of relevance, keyword frequency, and other ranking factors. Sophisticated indexing techniques use machine learning and artificial intelligence to maximize the precision of search results through context and intent analysis. This means that along with retrieving documents, indexing also needs to provide users with the most valuable and relevant content in the first place. As more sophisticated search algorithms emerge, indexing will keep improving to provide progressively superior search experience.

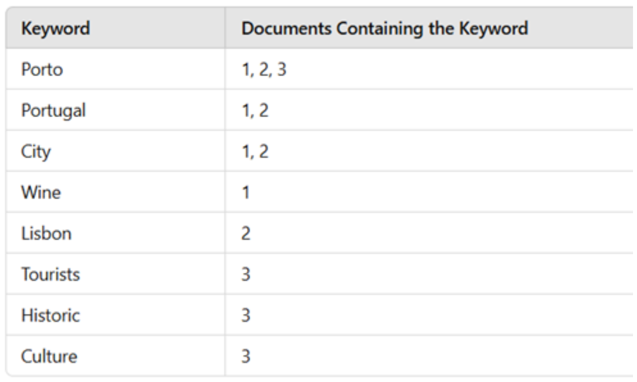
Also, indexing in an information retrieval system has a critical role in efficient searching and document retrieval. The collection of documents is the source of information, and extracted and processed terms are used by the indexer to create an inverted index. Systematic approach enables efficient document retrieval rapidly and accurately, and it is the basis of search technology today. In search engines, libraries, or business databases, indexing places massive volumes of information at individuals' fingertips and makes them available [9]. As information grows by the day, the future can only see more importance being placed on indexing, which will shape the future of accessing and retrieving information in a rapidly digitalizing world.

D. Example: Indexing in a Travel Website

To illustrate how indexing works, consider a travel website that stores information about different cities and countries. This website contains three documents with descriptions of various travel destinations. Document 1 states, *"Porto is a beautiful city in Portugal, known for its wine."* Document 2 describes, *"Portugal has many stunning cities, including Porto and Lisbon."* Finally, Document 3 explains, *"Tourists love visiting Porto for its historic architecture and vibrant culture."* These documents must undergo a series of steps to be indexed efficiently and facilitate fast and accurate search results.

- Document 1: "Porto is a beautiful city in Portugal, known for its wine."
- Document 2: "Portugal has many stunning cities, including Porto and Lisbon."
- Document 3: "Tourists love visiting Porto for its historic architecture and vibrant culture."

The system processes these documents, extracting important keywords and creating an inverted index, which links each word to the documents where it appears:



Keyword	Documents Containing the Keyword
Porto	1, 2, 3
Portugal	1, 2
City	1, 2
Wine	1
Lisbon	2
Tourists	3
Historic	3
Culture	3

Table 1. Example of Indexing process

Therefore, when the user queries "Porto," the system will search the inverted index directly and give back Document 1, Document 2, and Document 3—all the descriptions of Porto. When the user queries "Portugal," the system will give back Document 1 and Document 2.

Tokenization is the process of first indexation in which the document is broken down into words or terms. In processing, while the words are manipulated, they are manipulated without punctuation and also shifted to the lower case in view of facilitating similarity. For instance, the word "Porto" occurs in all three documents in a number of contexts, but when text is normalized to lowercase, the search system ensures that all instances of "porto" are treated identically.

Following tokenization, stopword elimination is done, wherein very common words that don't mean a lot—such as "is," "a," "in," "has," "for," and "and"—are eliminated. This filtering reduces the overall size of the index and allows search engines to assign greater significance to more important terms. Thus, the words "porto," "beautiful," "city," "portugal," "known," and "wine" remain from the first document, and similarly significant words are extracted from the second and third documents.

Once stop words are eliminated, words get stemmed and lemmatized to further optimize the indexing. Stemming takes words to their root form by stripping the suffixes, where a word like "cities" gets reduced to "city" and "visiting" gets reduced to "visit." Lemmatization takes words to their base dictionary form, thus making retrieval more efficient. After these habits are adopted, "tourists" turns into "tourist" and "architecture" turns into "architectur." These treatments avoid going into the minor spelling differences of the exact same word as separate entries in the search index.

From the short list of terms per document, therefore, an inverted index is constructed. An inverted index is a convenient data structure employed by information retrieval to associate each occurrence of a distinct term with documents in which it appears. Instead of storing documents themselves, the inverted index stores a list of words and an array of document IDs in which each occurrence of a word is. For example, the word "porto" across all three texts, the word "wine" in the first text only, and the word "lisbon" in the second alone only. This is far easier to search for in the system because it can go searching for similar documents by keyword instead of having to begin at the beginning of the full collection of documents.

Aside from simply transporting words and frequencies, search engines must also estimate the relevance of different terms in a document. It does this through term frequency-inverse document frequency (TF-IDF) weighting, where every term is assigned a number based on how frequently a term in a document and how infrequently a term is in all documents. Terms that appear very often in most of the documents, like "porto," are less weighted because they are frequent and contain less individual information. On the other hand, those terms that appear very rarely, like "wine" or "architecture," are given more weight, thereby making them more effective when ranking search results. This process of weighting serves to give priority to the most relevant documents when a user is looking for something.

Once the indexing is complete, the system is available for query processing and retrieval. If a user submits a search query for a keyword like "Portugal tourism," the search engine does not need to read whole documents but rather searches the inverted index with the most suitable entries. With the use of ranking algorithms like TF-IDF, the search engine orders documents with regard to relevance in such a way that relevant documents appear at the top of the search list.

V. MODELS

A. *BM25 in Web Search: A Detailed Information Retrieval Model*

In the era of very big Web search engines, information retrieval (IR) is playing a crucial role in determining the relevance of documents to user queries. The term-based Best Matching 25 (BM25) model is one of the most often used ranking functions in Web search and it is based on a probabilistic search framework. It shows improvements basing on previous models by using term frequency normalization and relevance score which is making it highly effective in ranking search results.

Modern Web search engines use BM25 as their main ranking function due to its efficiency, explainability, and strong theoretical basics. It is very useful in keyword-based search systems, where it assigns results to documents based on term frequency, inverse document frequency, and document length normalization. This chapter provides a detailed analysis of BM25, including its operation, mathematical basis, and performance in large-scale search systems.

In order to understand well the power and utility of BM25 that it has to offer, it is essential to understand its key concepts:

1) **Term Frequency (TF)**
   Term Frequency is referring to the number of times a term appears in a document. The idea is that the more frequently a term appears in a document, the more relevant that document is likely to be for a search query containing that term. But, BM25 is adjusting this frequency to prevent too high scores for documents with very high term frequencies.

2) **Inverse Document Frequency (IDF)**
   Inverse Document Frequency checks the importance of a term across the entire document collection. Terms that are common and appear across many documents are considered less significant, meanwhile rare terms are deemed more important. The IDF helps to penalize common terms and boosting rare ones.

3) **Document Length Normalization**
   Document Length Normalization adjusts the relevance score basing on the length of the document. The idea is that it prevents appearance bias towards longer documents, which may contain more instances of a term simply due to their length.

BM25 originates from the probabilistic information retrieval model, which assumes that the relevance of a document to a query can be estimated as a probability. Different than only Boolean models that rely on exact keyword matching, BM25 is ranking documents based on statistical relevance.

The probabilistic retrieval model is based on the Probability Ranking Principle (PRP), which states that documents should be ranked in descending order of their estimated probability of relevance to a user query.

Also, as stated in *The Probabilistic Relevance Framework: BM25 and Beyond* by Stephen Robertson and Hugo Zaragoza: “Whatever information is available, the system may make some statistical statement about the possible value of the relevance property. Given a binary document-by-document relevance property, then this statistical information may be completely encapsulated in a probability of relevance. The probability of relevance of a given document to a given query plays a central role in the present theory. We can in fact make a general statement about this: If retrieved documents are ordered by decreasing probability of relevance on the data available, then the system’s effectiveness is the best that can be obtained for the data. This is a statement of the Probability Ranking Principle (PRP).”

BM25 improves and refines this approach by using weighting mechanisms that balance term frequency (TF) and document length normalization. BM25 is a better version of usage the Probability Ranking Principle by ensuring that frequently occurring terms in a document do not disproportionately boost its relevance score. The model adjusts for document length so that longer documents are not unfairly advantaged simply because they contain more terms. BM25 introduces a saturation effect, meaning that after a certain threshold, additional occurrences of a term contribute less to the ranking score. The model also applies inverse document frequency (IDF) to prioritize rare but meaningful terms, making it more effective at distinguishing relevant documents. As a result, BM25 creates a more refined ranking system that balances term importance and document properties to improve search result accuracy.

The Probability Ranking Principle (PRP) states that the optimal ranking of documents is achieved by ordering them in decreasing probability of relevance P(R=1 | d, q) , where:

$$P(R=1|d,q) = \frac{P(d|R=1)P(R=1)}{P(d|R=1)P(R=1)+P(d|R=0)P(R=0)}$$

Applying Bayes' theorem and assuming that query terms are conditionally independent given relevance, the odds ratio is:

$$\frac{P(R=1|d,q)}{P(R=0|d,q)} = \frac{P(R=1)}{P(R=0)}\prod_{i=1}^{n}\frac{P(q_i|R=1)}{P(q_i|R=0)}$$

Taking the logarithm:

$$\log P(R=1|d,q) \propto \sum_{i=1}^{n} log\frac{P(q_1|R=1)}{P(q_1|R=0)}$$

Term Frequency Adjustment

Instead of using raw term frequency $TF(q_i,d)$, BM25 applies a saturation function to prevent excessive weighting of frequent terms. This is modeled as:

$$TF'(q_i,d) = \frac{TF(q_i,d)(k_1+1)}{TF(q_i,d)+k_1}$$

where k_1 is a tunable parameter controlling the saturation effect.

Inverse Document Frequency (IDF) accounts for term importance across the collection. It is defined as:

$$IDF(q_i) = log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

Where:

- N is the total number of documents,

- $n(q_i)$ is the number of documents containing term q_i.

Document Length Normalization

$$TF''(q_i, d) = \frac{TF'(q_i, d)}{1 - b + b \cdot \frac{|d|}{avg_d}}$$

where b is a parameter (typically set around 0.75) that controls the extent of length normalization, d is document length and avg_d is average document length.

Combining all components, the final BM25 ranking function is:

$$BM25(d, q) = \sum_{i=1}^{n} IDF(q_i) \times \frac{TF(q_i, d) \cdot (k_1 + 1)}{TF(q_i, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avg_d})}$$

The BM25 ranking function works by calculating a score for each pair of document-query based on various factors such as term frequency, inverse document frequency, and document length normalization. The steps below are describing the structure of the algorithm with the application of the BM25 formula. The process contains the following steps:

1. Extraction of the query and the set of documents to be ranked.

2. Then calculating term frequency (TF) for the query terms in each document.

3. Calculating inverse document frequency (IDF) for each query term.

4. Application of the BM25 formula to compute the score for each document.

5. Ranking the documents based on their BM25 scores.

The BM25 algorithm starts by calculation of the relevance of a document for a given query. This is done by using a probabilistic approach where the score is based on the term frequency in the document and the inverse document frequency of the query terms across the entire collection. The BM25 formula uses term frequency matrix denoted as (TF) to check how often a term appears in a document and inverse document frequency matrix denoted as (IDF) and how rare or common a term is across all documents.

In BM25, the (TF) can be adjusted by using two parameters: k1 (which controls the sensitivity to term frequency) and b (which adjusts for document length). These parameters help with fine-tuning the contribution of TF and document length normalization to the calculated final relevance score.

While BM25 ranking function does rely on a probabilistic model of relevance, it does not explicitly Bayesian inversions as part of its ranking mechanism. Instead, BM25 assumes a much simpler probabilistic framework based on the conditional independence of terms. What each term contributes the relevance score is treated independently, which is making the model computationally efficient, though it relies on the assumption that terms are conditionally independent, given the document relevance.

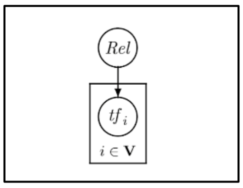


Figure V-1: Graphical model indicating basic independence assumptions, source: [20]

Despite some mentioned simplifications, the BM25 algorithm is very effective for ranking documents in real-world information retrieval systems. The final score combines the effects of text characteristics embedded in term frequency matrix, inverse document frequency matrix, and document length normalization. As a result, ranking system that reflects both the importance of query terms and the relative lengths of the documents has managed to be.

There are two main tuning parameters in BM25, $k_1$and b. Their values have significant impact on the effectiveness of ranking and should be adjusted based on the characteristics of the document collection:

- k_1 - in other words Term Frequency Saturation Control. We use it to determine how much term frequency affects the score. A lower value is meant to reduce the impact of term frequency, while a higher value increases its effect.

- b - which is Length Normalization Factor. This factor controls to what extent document length normalization is applied. A value of 0 means no length normalization, while 1 uses full normalization.

In web search, very popular values are $k_1 \approx$ 1.2 - 2.0 and $b \approx 0.75$, which usually provide a good compromise between sensitivity of term frequency and length normalization.

In these times that we have now modern search engines are tasked with indexing billions of documents and that requires very efficient ranking models. BM25 stays a preferred choice thanks to its computational efficiency. Very big scale search engines optimize BM25 by:

- Usage of Inverted Indexes: Instead of scanning entire documents, search engines store term frequencies in inverted indexes, allowing fast retrieval.
- Precomputation of IDF Scores: Because IDF values remain constant for a given document corpus, they can be easily precomputed and cached.
- Optimized Data Structures: Storing everything in a compressed format reduces memory requirements and efficient lookup tables reduces computational overhead.
- BM25-Adaptive: Its main characteristic is that it dynamically adjusts k1 and b parameters based on query characteristics in order to optimize relevance ranking.

These and many other optimizations let BM25 scale effectively for high-traffic search engines such as for example Google, Bing, and Elasticsearch.

Real-World Applications

- Showing BM25 as a Cost-effective Semantic Cache

When implementing LLMs into systems of production, the high API costs associated with frequent LLM queries can become a challenge. Using vector databases (VecDBs) as a semantic cache and combining them with BM25-based retrieval, we can achieve cost-effective end-to-end LLM applications.

- Handling Hallucinations in LLMs

One of the biggest challenges with pure LLMs is their tendency to generate plausible but incorrect information in the factual context, known as hallucinations. By incorporating BM25-based retrieval as a component in the overall system, the outputs can be explained on ground of actual document content, reducing the risk of hallucinations.

Challenges and Considerations

The key challenges and considerations in applying the BM25 ranking algorithm in real-world applications include:

- Handling Long Documents

The search results indicate that BM25 tends to overly penalize very long documents, as the document length normalization component can become too dominant. This can cause relevant longer document content being ranked lower than it really should be. Addressing this issue may require changes to the BM25 formula or the hybrid approaches that combine BM25 with other ranking signals.

- None of Semantic Understanding

As stated, BM25 is a term-based ranking algorithm that does not capture the semantic meaning of queries and documents. When dealing with queries that require a deeper understanding of context this can become a limitation.

B. ***Dense Passage Retrieval***

Historical approaches such as TF-IDF and BM25 search for words that match but struggle with grasping meaning. With the emerging advancements in neural information retrieval, Dense Passage Retrieval (DPR) emerged as a deep learning approach which is capable of retrieving documents on the basis of meaning rather than word matching. It maps queries and documents to a more compact, dense space so they can be accessed efficiently with Approximate Nearest Neighbor (ANN) search techniques. DPR is trained to figure out what a query and document are about, whereas BM25 scores documents in terms of the frequency of occurrence of terms. Due to this, DPR excels at returning synonyms, reformulated questions, and complex search requirements [12], [13].

DPR plays a critical role during the retrieval stage of a web search. This stage identifies documents that best match the user's desire with speed. Following that, there is a ranking stage that ranks the documents once more with more robust models such as cross-encoders.

Architectural Components of DPR

The architecture of DPR model includes two neural network encoders (bi-encoder architecture), a method of measuring similarity, and an effective method of retrieval. Below are the specifics of the components.

a) Bi-Encoder Architecture

DPR employs a bi-encoder architecture where two independent encoders are trained:

- Query Encoder : Maps a user query into a dense vector representation.

- Document Encoder : Maps candidate documents into dense vector representations.

Each encoder is implemented as a deep neural network, typically a pretrained BERT model [14].

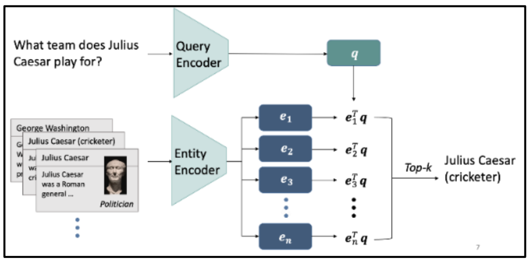


Figure V-2: Example of a Bi-Encoder, source: [15]

b) The BERT Architecture

Bidirectional Encoder Representations from Transformers (BERT) is a deep bidirectional Transformer, i.e., it processes text in both left-to-right and right-to-left directions simultaneously, as opposed to uni-directional ancestors. Being bidirectional allows BERT to learn more contextual word relationships, allowing it to better understand search queries and document semantics [16], [17].

Transformer-Based Neural Network

BERT is built on the Transformer architecture, using the following techniques:

- Self-Attention Mechanism: Enables the model to focus on different parts of a text dynamically, allowing better contextual understanding.
- Multi-Layered Encoding: Uses multiple Transformer layers (e.g., 12 for BERT-Base, 24 for BERT-Large) to learn deep representations.
- Positional Embeddings: Incorporates word order information to distinguish different meanings of a sentence.

Pretraining Strategy

BERT is pretrained using two self-supervised learning tasks. The first one is Masked Language Model (MLM). This randomly masks words in a sentence and learns to predict them based on surrounding words. Besides this, BERT also uses Next Sentence Prediction (NSP), which trains the model to predict whether one sentence logically follows another, improving coherence understanding.
This pretraining enables BERT to comprehend many various search queries and texts.

c) Applications of BERT in Web Search

BERT-based models find application in various aspects of contemporary search engines, such as comprehending questions, searching for texts, and ranking them.

Query Understanding and Expansion

One of the most critical challenges in web search is interpreting ambiguous queries. Conventional search engines primarily rely on keywords to produce results, which can result in poor results if the user intent is ambiguous. BERT enhances question comprehension in these ways:

- Disambiguating polysemous words (e.g., "Apple" as a company vs. fruit).
- Handling conversational and long-tail queries with complex intent.
- Generating query reformulations by expanding queries with semantically related terms.

For example, in Google Search, BERT helps interpret queries like:

- *"Can you get medicine for someone from a pharmacy?"*

A non-BERT model might match documents discussing "getting medicine" in general, whereas BERT understands the underlying legal and procedural aspects.

Passage Ranking with Cross-Encoders

In the ranking step, cross-encoders based on BERT are usually employed to determine how applicable a document is to a question. Cross-encoders process the document and question as a single input collectively, enabling them to interact better:

The query and document are concatenated as:

$$[CLS]\ Query\ [SEP]\ Document\ [SEP]$$

Where $[CLS]$ is the classification token and $[SEP]$ is the separator token. BERT then processes this input, and the final [CLS] token embedding is used to predict a relevance score. Unlike retrieval-stage models, cross-encoders do not precompute embeddings, making them computationally expensive but highly accurate.
This approach ensures that search results are ranked by meaning rather than matching words alone, significantly enhancing search quality.

d) Query and Document Encoding

Both encoders share the same architecture and employ a Transformer-based approach. The input text is split into tokens and goes through self-attention layers, which generate a contextualized embedding. The output vector is derived from the [CLS] token, which represents the entire input.
For a given query q_{vec} and a document d_{vec}, the respective encoders output:

$$q_{vec} = f_q(q) \in \mathbb{R}^d$$
$$d_{vec} = f_d(d) \in \mathbb{R}^d$$

where *d is* the embedding dimension.

e) Similarity Computation

DPR ranks documents based on the dot product (or cosine similarity) between the query embedding and document embeddings:

$$score(q, d) = q_{vec} \cdot d_{vec}$$

This step ensures that documents with embeddings closer to the query in vector space are retrieved first.

Training Methodology

- Supervised Contrastive Learning

DPR is trained using contrastive learning, where the model learns to distinguish relevant documents from irrelevant ones. Given a query, three types of document representations are used:

1. Positive sample : A document relevant to the query.
2. Negative sample : A document irrelevant to the query.
3. Hard negatives: Documents that are lexically similar but not semantically relevant.

The objective of training is to bring the good document and the query together closer and push them apart from bad documents. This is achieved through the application of the Negative Log Likelihood (NLL) loss:

$$L = -\log \frac{e^{q_{vec} \cdot d_{vec}^{+}}}{e^{q_{vec} \cdot d_{vec}^{+}} + \sum_j e^{q_{vec} \cdot d_{vec,j}^{-}}}$$

where the denominator includes hard negatives, making training more robust.

Efficient Retrieval and Indexing

a) Precomputing Document Embeddings

One of the major advantages of DPR is that document embeddings can be precomputed and stored in an ANN (Approximate Nearest Neighbor) index. By doing so, the efficiency of the model increases, since only the query must be converted into a code during searching.

b) Approximate Nearest Neighbor (ANN) Search

DPR employs dense vector similarity, and as such, it is not feasible to browse millions of documents directly. Due to this, ANN search algorithms such as FAISS (Facebook AI Similarity Search) and HNSW (Hierarchical Navigable Small World graphs) are employed to efficiently retrieve the top-k most similar document embeddings.

The steps of the retrieval are as follows:

1. The query is encoded using the query encoder .
2. The dense vector is matched against the recomputed document index using ANN search.
3. The top k-nearest neighbor documents are retrieved.

This system supports fast and large-scale document search that is suitable for real-world web search.

Conclusion

Dense Passage Retrieval (DPR) is a significant improvement in information retrieval since it learns to comprehend and capture meaning using deep learning. It's bi-encoder architecture, contrastive loss approach, and efficient ANN-based retrieval make it search through vast volumes of information on the web in a matter of seconds. DPR is never used in isolation, though; it performs optimally in combination with conventional retrieval techniques and cross-encoder ranking models in a multi-stage search pipeline, which is discussed in the next section.

*C. **Hybrid Models***

Modern web search systems must cope with billions of pages and the ambiguous nature of natural language. Traditional approaches such as BM25 are able to identify potential matches in a quick manner by employing word frequency statistics and precise information in documents. Dense retrieval models such as DPR, which employ BERT, learn deeper semantic meanings through sophisticated vector representations. Although both approaches have their strengths, hybrid combinations are gaining popularity to eliminate their deficiencies. In this section, we discuss the technical aspects of hybrid models that blend BM25 and BERT-based DPR, such as fusion techniques, system integration, and real-world issues in employing these systems for web search [18], [19].

Hybrid Model Architecture

The hybrid approach we are considering employs BM25 to identify potential candidates in the initial step and BERT-based DPR to rank them once more. As we have already discussed how BM25, DPR, and BERT operate, we are now concerned with how these components can be effectively combined.

a) Two-Stage Retrieval Pipeline

The hybrid approach is implemented in a two-stage retrieval pipeline:

1. Candidate Generation Stage: BM25 is employed to rapidly retrieve a broad set of candidate documents from a large-scale inverted index. Its efficiency enables processing of extensive document collections with minimal computational overhead.
2. Semantic Re-ranking Stage**:** The candidate set is then passed to a BERT-based DPR model, which re-ranks the documents based on dense semantic representations. This re-ranking leverages the rich contextual embeddings produced by BERT to assess the true semantic relevance of each candidate to the query.

This incremental approach enables the system to apply BM25 to locate a large number of relevant documents rapidly. It then applies a more sophisticated model to a smaller set of higher-quality documents.

b) Fusion Strategies

There are good hybrid approaches that combine various scoring approaches from BM25 and dense retrieval. We consider three primary approaches to achieve this:

Combining Scores

Score fusion is where we add the BM25 score and the dense model score into a single ranking number. Since the two scores are from different data sets, they must be normalized. The common method of doing this is to apply min-max normalization or Z-score normalization to each score, then combine them with weights.

$$s_{hybrid} = \alpha \cdot \tilde{s}_{BM25}(q,d) + (1-\alpha) \cdot \tilde{s}_{DENSE}(q,d)$$

Here, α is a hyperparameter that adjusts the relative influence of the BM25 and dense scores. This strategy benefits from the simplicity of linear combination, allowing for tunable balance between recall and precision.

Cascading

In cascading, BM25 provides the initial list of candidates. The BERT-based DPR re-ranker then provides new relevance scores. Rather than merging scores, the dense model re-ranks the list produced by BM25. This approach is easier to merge scores because the two processes are distinct. Yet it requires delicate tuning of the candidate set size to ensure the dense model has sufficient context to re-rank without inducing excessive delay.

Learning-to-Rank Approaches

A superior blending approach employs a learning-to-rank (LTR) method. In this approach, a second ranking model is trained on BM25 and dense model features. These features may be the BM25 score, dense similarity score, query length, and other document information. The LTR model, typically a gradient-boosted decision tree or a neural network, learns to provide a final ranking score by optimizing a ranking loss using labeled relevance data. This approach can learn intricate relationships between features and may enhance ranking performance, but it requires more training data and effort.

Conclusion

Hybrid retrieval systems that combine BM25 with BERT-based DPR models are an effective method of enhancing web search results. By combining the fast and efficient candidate generation of BM25 with the deep semantic understanding of dense retrieval models, these systems are able to produce improved search results in terms of recall and precision. This paper examines the technical aspects of combining hybrid models, including how to combine them, scoring normalization, indexing issues, and training issues. As web search continues to evolve, the construction and development of hybrid systems will be critical to address the increasing demands of information retrieval.

*D. **Ranking Functions Cross-Comparison from BM25, VSM, and Neural Retrieval Models***

a) BM25: Probabilistic Ranking Function

BM25 (Best Matching 25) is a probabilistic ranking function based on the probabilistic information retrieval framework. It assesses documents relevance based on term frequency short (TF), inverse document frequency short (IDF), and document length normalization. BM25 ranking function is given by:

$$BM25(d,q) = \sum_{i=1}^{n} IDF(q_i) \times \frac{TF(q_i,d) \cdot (k_1+1)}{TF(q_i,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avg_d})}$$

Where:

- $TF(q_i, d)$ is term frequency of term q_i in document d.
- $IDF(q_i)$ is the inverse document frequency of term q_i.
- k_1, b are changeable hyperparameters that control term frequency scaling and document length normalization.
- $|d|$, avg_d are document length and average document length

Advantages of BM25 are:

- it is very effective for keyword-based searches.
- it is very efficient and computationally inexpensive.
- ranking scores are very easy to explain.

Weaknesses:

- BM25 is lacking semantic understanding of the queries
- The model is assuming independence of query terms, it is limiting its ability to capture term relationships.

b) Ranking of Vector Space Model (VSM) and Cosine Similarity

VSM is representing queries and documents by using term-weighted vectors in a multidimensional matrix and ranks documents based on cosine similarity in these spaces. It is given by:

$$Sim(d,q) = \frac{\sum_{t \in V} w_{t,d} \cdot w_{t,q}}{\|d\| \cdot \|q\|}$$

Where the variables:

- $w_{t,d}$ and $w_{t,q}$ are the term weights in the document and query vectors.
- $\|d\|$ and $\|q\|$ are magnitudes of a given vector.

Advantages:

- It can capture term importance using term frequency-inverse document frequency (TF-IDF).
- It is a simple and interpretable ranking method.

Weaknesses:

- It has a similar assumption as BM25 assumes independence of query terms.
- It does not account for term order or contextual meaning.
- It is computationally expensive for big-scale search applications.

c) Neural Retrieval Models: Deep Semantic Matching

Neural retrieval models, such as BERT-based retrieval and Dense Passage Retrieval (DPR), are using deep learning to compute ranking scores that are based on semantic understanding different than vector space models.

a. BERT-based Retrieval Ranking Function

BERT-based models rank documents using contextual characteristics rather than by matching terms. The ranking score is computed using a neural scoring function which is given by:

$$S(d,q) = f(BERT(q,d))$$

Where:

- $BERT(q,d)$ is a variable that is representing the joint correlation of query and document.
- f () is a neural network that is used to predict the relevance score.

Strengths:

- It captures deep semantic relationships and contextual meaning.
- It is able to handle word polysemy and synonyms in an effective way..
- It also can generalize to diverse query patterns.

Disadvantages:

- From computational standpoint it is very expensive compared to traditional models.
- Also it requires well prepared meaningful and well labeled training data.

b. Dense Passage Retrieval (DPR) Ranking Function

DPR maps queries and documents into a dense vector space and retrieves documents via nearest neighbor search. The ranking function is:

$$S(d,q) = cosine(E_q(q), E_d(d))$$

Where:

- $E_q(q)$ and $E_d(d)$ are query and document embeddings.

Advantages:

- It is able to scale well with precomputed embeddings and Approximate Nearest Neighbor (ANN) search.
- It learns valuable representations beyond keyword matching.

Weaknesses:

- It is still requiring significant training data, same as the model before.
- It has low explainability.

d) Hybrid Retrieval: Combining BM25 and Neural Models

Hybrid retrieval approaches usually combine BM25 model and neural ranking models to achieve both lexical matching and semantic understanding. A very common hybrid ranking function is given by:

$$S_h(d,q) = \alpha \cdot S_{BM25}(d,q) + (1-\alpha) \cdot S_{neural}(d,q)$$

Where α is a variable that balances traditional term-based ranking and neural ranking.

Strengths:

- By leveraging both term-based and semantic ranking it significantly improves recall.
- It filters retrieved documents well thanks to using BM25 before hybrid model applies computationally expensive neural ranking.

Weaknesses:

- It is challenging to tune for different datasets.
- It is still computationally demanding due to neural components.

VI. Comparison of the Models

A detailed comparison of the models in tabular form can be seen in *Annex A - Comparison Table.*

The plots below are based on an experiment conducted using the TREC (Text Retrieval Conference) dataset. This dataset is widely used for checking information retrieval systems. The experiments aim was to evaluate the performance of different retrieval models such as BM25, Vector Space Model (VSM), BERT-based retrieval, Dense Passage Retrieval (DPR), and Hybrid Retrieval approaches(the model discussed above). The performance metrics that were used are—Precision, Recall, and F1 Score— and they were computed for each model based on their ability to retrieve relevant documents from a large corpus. The results are presented in the plots and offer a comparative analysis of these models in terms of their effectiveness in accurately retrieving relevant documents and at the same time while minimizing false positives and negatives. The results also provide insights into their strengths and weaknesses in real-world retrieval tasks.

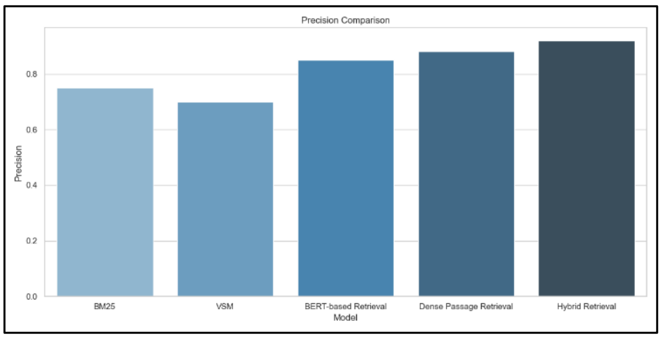


Figure VII-1: Comparison of precision for different models, source: [22]

- BM25 retrieves highly relevant documents but still has some false positives – precision: 0.75.
- VSM performs decently, but documents retrieved might not always match the query intent – precision: 0.70.
- BERT is excellent at understanding the context and delivering highly relevant documents – precision: 0.85.
- DPR performs well in returning highly relevant results based on semantic embeddings – precision: 0.88.
- The hybrid model combines the efficiency of BM25 with the semantic understanding of neural models, leading to very high precision – precision: 0.92.

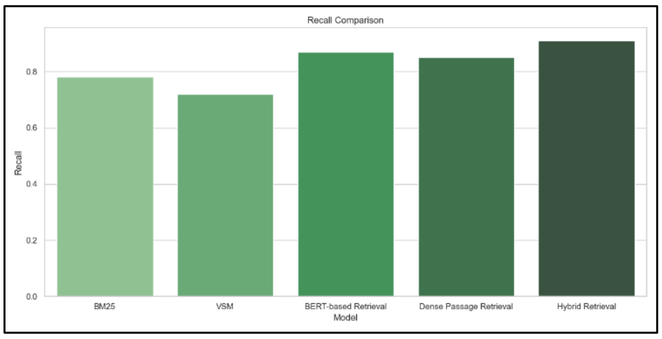


Figure VII-I1: Comparison of recall for different models, source: [22]

- BM25 retrieves most relevant documents but may miss a few – recall 0.78.
- VSM retrieves most relevant documents but may miss others due to lack of semantic understanding – recall: 0.72.
- BERT tends to capture most relevant documents due to its ability to understand the full meaning – recall: 0.87.
- DPR retrieves most relevant documents but might miss out on some edge cases where the semantic embedding isn't as strong – recall: 0.85.
- Hybrid model captures a very high number of relevant documents while avoiding many false negatives – recall: 0.91.

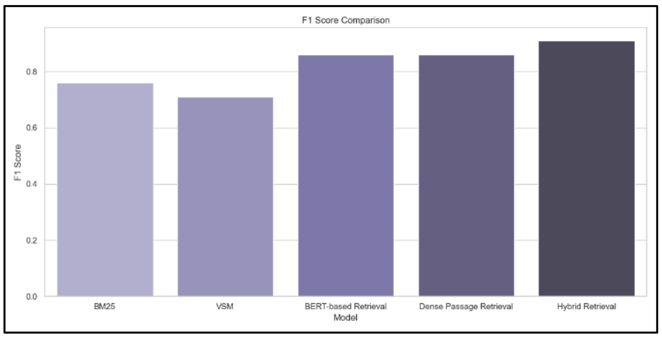


Figure VII-1II: Comparison of F1-score for different models, source: [22]

- BM25 performs well in balance, though not perfect – F1 - score: 0.76.
- VSM model is not bad, but not optimal – F1 - score: 0.71.
- BERT shows strong balance between precision and recall – F1 – score: 0.86.
- For DPR - again, a solid F1 score shows good balance between precision and recall – F1 - score: 0.86.
- High F1 score shows that the hybrid model balances precision and recall exceptionally well – F1 - score: 0.91.

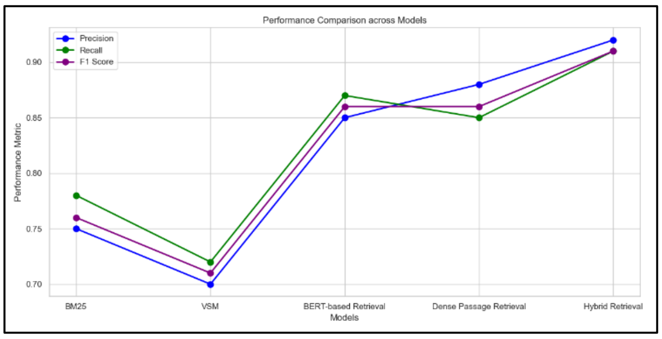


Figure VII-1V: Performance comparison for different models, source: [22]

Final analysis:

- These values definitely show that BM25 is effective for search tasks especially ones that rely heavily on matching query terms with document terms. However, it can be less effective when deeper semantic understanding is needed.

- VSM is prone to struggling with understanding the nuances of language, for example synonyms or polysemy. Secondly, it works on term-matching, which can lead to lower precision and recall compared to more advanced models.

- The higher performance of BERT-based retrieval comes from its ability to understand context, synonyms. Additionally, the broader meaning of terms in queries and documents. It handles more complex queries better than traditional models like BM25 or VSM but at a cost of computational overload.

- Dense Passage Retrieval models generally perform better than traditional models (like BM25) in tasks that require semantic understanding. Answering questions or retrieving contextually relevant information are tasks on which the model achieves better results than BM25.

- Concluding, hybrid retrieval systems combine the computational efficiency of BM25 with the semantic power of neural models. At the same time achieving superior performance over either individual model alone. This approach works well in complex search tasks like question answering or retrieving documents based on more abstract queries.

VII. Conclusion

This seminar has been an overview of the interesting and complex topic of web searching and web-based information retrieval. As our cyber space expands, the ability to efficiently locate and collect useful, credible information more than ever is critical. For the general web user, researchers, and businesses, search engines become the hub of how we find and interpret large volumes of web-based information.

Among the prominent issues of debate was how search engines work nowadays, including the role of indexing, ranking algorithms, and retrieval models in bringing the most pertinent results to users. We discussed how search engines like Google and Bing continue to develop ranking systems with artificial intelligence, machine learning, and natural language processing in order to enhance accuracy and user satisfaction. These mechanisms need to be understood as they not only dictate what is remembered but also how it feels and how it is conveyed.

The second important topic discussed in this seminar was search effectiveness with credibility. The internet is full of misinformation, biased sources, and poor quality sources, and therefore the technology employed to search needs to be directed to credible and reliable sources. The topic drew the search engine's end-users and developers analyzing the sources critically, monitoring search bias, and knowing the limitations of algorithmic content retrieval.

Lastly, web search and information retrieval are not so much a question of making information available but, instead, a question of how knowledge is transmitted and received in the modern age. This seminar has given us penetrating insights into the mechanics, challenges, and ethical implications of search technology. In the future, search engine result gravity, conscientiousness, and a demand for transparency of search algorithms will be the benchmarks of offering information present, equal to all, and reliable.

References

[1] SEO.com, "How search engines work: Crawling," [Online]. Available: https://www.seo.com/basics/how-search-engines-work/crawling/

[2] R. van Veen, "Crawling, indexing and ranking: The meanings and differences," [Online]. Available: https://ralfvanveen.com/en/technical-seo/crawling-indexing-and-ranking-the-meanings-differences/

[3] Wikipedia, "Web crawler," [Online]. Available: https://en.wikipedia.org/wiki/Web_crawler

[4] Cloudflare, "What is a web crawler?" [Online]. Available: https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/?utm_source=chatgpt.com

[5] Akamai Technologies, "What is a web crawler," [Online]. Available: https://www.akamai.com/glossary/what-is-a-web-crawler

[6] S. Büttcher, C. Clarke, and G. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines*, Cambridge, MA: MIT Press, 2010. [Online]. Available: https://books.google.pt/books?id=SKzBBAAAQBAJ

[7] M. A. A. Mamun, M. S. Islam, and M. R. Islam, "Information retrieval systems: A methodological review," [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-73125-9_36

[8] A. K. Maurya and A. K. Tripathi, "A review on recent research in information retrieval," [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922005191

[9] Fagbola, O., "Indexing and abstracting as tools for information retrieval in digital libraries," [Online]. Available: https://www.sciencegate.app/document/10.4018/978-1-5225-3914-8.ch027

[10] O. A. Abass and O. A. Arowolo, "Comprehensive study and comparison of information retrieval indexing techniques," [Online].

Available: https://thesai.org/Downloads/Volume7No1/Paper_20-Comprehensive_Study_and_Comparison_of_Information_Retrieval_Indexing_Techniques.pdf

[11] LinkDoctor.io, "Crawler Directives: How to Guide Web Crawlers," [Online]. Available: https://linkdoctor.io/crawler-directives/

[12] V. Karpukhin *et al.*, "Dense Passage Retrieval for Open-Domain Question Answering." [Online]. Available: https://github.com/facebookresearch/DPR.

[13] X. Ma, K. Sun, R. Pradeep, J. Lin, and D. R. Cheriton, "A Replication Study of Dense Passage Retriever," 2021. [Online]. Available: http://pyserini.io/

[14] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." [Online]. Available: https://github.com/tensorflow/tensor2tensor

[15] "TABi: Type-Aware Bi-Encoders for Open-Domain Entity Retrieval · Hazy Research." Accessed: Mar. 29, 2025. [Online]. Available: https://hazyresearch.stanford.edu/blog/2022-04-19-contrastive-3

[16] R. Gupta, "Bidirectional encoders to state-of-the-art: a review of BERT and its transformative impact on natural language processing," *Информатика. Экономика. Управление - Informatics. Economics. Management*, vol. 3, no. 1, pp. 0311–0320, Mar. 2024, doi: 10.47813/2782-5280-2024-3-1-0311-0320.

[17] J. Wang *et al.*, "Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges," Feb. 2024

[18] J. Rayo, R. De La Rosa, and M. Garrido, "A Hybrid Approach to Information Retrieval and Answer Generation for Regulatory Texts".

[19] "Hybrid Document Retrieval | Haystack." Accessed: Mar. 30, 2025. [Online]. Available: https://haystack.deepset.ai/blog/hybrid-retrieval

[20] Robertson, Stephen, and Hugo Zaragoza. "The probabilistic relevance framework: BM25 and beyond." Foundations and Trends® in Information Retrieval 3.4 (2009): 333-389.

[21] Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations andTrends® in Information Retrieval, 3(4), 333-389.

[22] Pannu, Mandeep & James, Anne & Bird, Robert. (2014). A Comparison of Information Retrieval Models. Proceedings of WCCCE 2014: The 19th Western Canadian Conference on Computing Education - In-Cooperation with ACM SIGCSE. 10.1145/2597959.2597978.

[23] Manal Sheikh Oghli , Muhammad Mazen Almustafa, 2021, Comparison of basic Information Retrieval Models, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 10, Issue 09 (September 2021

[24] Nuno Escudeiro, Ricardo Almeida, Information Retrieval and Text Mining – Introduction [Online]. Accessed: Mar. 30, 2025. Available: https://moodle.isep.ipp.pt/pluginfile.php/441463/mod_resource/content/5/t1-2%20introduction.pdf

ANNEXES:

A – Comparison Table

Retrieval Model	Matching Type	Description	Real World Applications	Pros	Cons
BM25 (Best Matching 25)	Lexical	A probabilistic ranking function using term frequency (TF) and inverse document frequency (IDF).	Search engines (Google Scholar, legal research), e-commerce search.	Fast, interpretable,explainable, works well on short queries and effective for keyword search	Limited semantic understanding, for example - struggles with synonyms and semantic meaning.
Vector Space Model (VSM)	Lexical	Represents queries and documents as vectors in a multi-dimensional space, using cosine similarity.	Classic information retrieval systems, library search, document ranking.	Simple and intuitive, works well for small-scale applications, well captures term importance	Doesn't handle synonymy or word variations well. In other words does not account for term order or meaning
Neural Retrieval (Deep Semantic Matching)	Semantic	Uses deep learning models to understand query-document relationships at the semantic level.	AI (ChatGPT search), recommender systems, customer support.	Deep contextual understanding, robust to synonyms as result handles synonyms and contextual meaning well.	Computationally expensive, requires large labeled training data.
Dense Passage Retrieval (DPR) Ranking Function	Semantic	Uses dense embeddings to retrieve relevant passages based on deep learning models.	Open-domain QA (e.g., Wikipedia retrieval), biomedical search.	Effective for complex, long-tail queries, high precision. Learns dense representations, scalable with ANN search	Requires extensive training and large storage for embeddings. Less explainable/interpretable.
Hybrid Retrieval (BM25 + Neural Models)	Mixed	Combines BM25's lexical matching with neural retrieval for improved ranking.	Enterprise search (e.g., corporate document retrieval), academic research.	Balances lexical and semantic retrieval. Best of both worlds: speed of BM25 + semantics of neural models.	Requires more infrastructure and tuning, for example, it is hard to optimise weighting factor