Team 3DS2A at LongEval: Performance Evaluation over Time of IR Systems with Proximity Search and Reranking Components

Notebook for the LongEval Lab at CLEF 2025

Andrea Bruttomesso[1,*,†], Daniele Cavazza[1,*,†], Alessandro Corrò[1,*,†], Simone Peraro[1,*,†], Davide Seghetto[1,*,†] and Nicola Ferro[1,*,†]

[1]University of Padua, Via 8 Febbraio, 2 - 35122 Padova, Italy

Abstract

This report gives an overview of the system developed by Team 3DS2A for Task 1 of the LongEval Lab at CLEF 2025. The team members are students enrolled in the Computer Engineering master's program at the University of Padua. The team developed an information retrieval system tailored to run on a French-language document corpus. The system was evaluated over a nine-month timespan to assess its robustness in terms of precision and recall over time. Throughout development, multiple techniques were explored, including alternative text analyzers, proximity search, chunk-based indexing, and semantic reranking using sentence embeddings. This report presents the system architecture, the experimental strategies adopted, and the performance achieved during the training phase.

Keywords

CLEF, LongEval 2025, Information Retrieval, Search Engine, Reranking, Word Embeddings

1. Introduction

In today's digital age, online searching has become an integral part of daily life, with billions of users worldwide relying on search engines to quickly and accurately find the information they need. A search engine (SE) is a software system designed to fulfill this demand by processing user queries that express specific information needs. However, as the number of web pages continues to grow rapidly, search engines face increasing challenges, one of the most significant being a decline in performance over time. To address this issue, the LongEval Lab, organized by the Conference and Labs of the Evaluation Forum (CLEF), encourages the development of temporal information retrieval (IR) systems capable of adapting to the evolving nature of online content [1]. This work reports on the proposed solution implemented by the 3DS2A group at the University of Padua as part of the Search Engines Course.

Alongside the traditional search pipeline, the system also utilizes more sophisticated techniques like chunk-based search, pseudo-relevance feedback, and reranking based on a sentence embeddings model to enhance the retrieval capability of the queries used.

The paper is organized as follows: Section 2 briefly introduces some related works for past LongEval tasks at CLEF 24; Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings; Section 6 presents the statistical analysis performed to assess the significance of performance differences between systems; finally, Section 7 draws some conclusions and outlooks for future work.

CLEF 2025 Working Notes, 9 - 12 September 2025, Madrid, Spain

*Corresponding authors.

†These authors contributed equally.

andrea.bruttomesso.1@studenti.unipd.it (A. Bruttomesso); daniele.cavazza@studenti.unipd.it (D. Cavazza); alessandro.corro.1@studenti.unipd.it (A. Corrò); simone.peraro.1@studenti.unipd.it (S. Peraro); davide.seghetto@studenti.unipd.it (D. Seghetto); nicola.ferro@unipd.it (N. Ferro)

https://www.dei.unipd.it/~ferro/ (N. Ferro)

0009-0005-8958-5459 (D. Cavazza); 0009-0001-2760-685X (S. Peraro); 0000-0001-9219-6239 (N. Ferro)

2. Related Work

The LongEval 2025 task follows the same general structure as previous CLEF LongEval editions, with the key difference that this year's training set spans a significantly longer temporal window.

In past editions, a wide range of information retrieval techniques have been explored, with sentence embedding-based approaches playing a prominent role, especially in the re-ranking phase [2] [3]. For example, Basaglia et al. [4] adopted a reranking strategy based on sentence embeddings, demonstrating its effectiveness in improving retrieval performance on temporally dynamic datasets.

While the discussion about whether stemming or lemmatizing achieves better performance is still open [5] [6], another common technique involves the expansion of query terms [7], and the use of Named Entity Recognition [8].

Our objective is to explore some of these techniques, combining them into new approaches while participating at the LongEval Web retrieval task.

3. Methodology

This section describes the methodologies adopted by our IR system for this task. To build our IR system, we used Apache Lucene 10.1.0 [9], an open source search engine library developed in Java, known for its high performance and rich feature set. Lucene provides a robust framework for indexing and retrieval, which we extended with a reranking module based on sentence embeddings.

Our goal was to enhance the effectiveness of an approach explored in the previous year by Basaglia et al. [4]. We used a similar approach based on reranking a small chunk of documents, but using a larger and more expressive sentence embedding model to capture deeper semantic similarities between queries and documents.

The main components of our system are:

- **WebDocumentParser** to read and parse a *.json* file into a **WebDocument** object.
- **WebAnalyzer** to analyze the contents of a document and return tokens ready to be indexed.
- **DirectoryIndexer** to manage the opening, parsing and indexing of the contents in a specific directory.
- **Searcher** to load the provided queries and perform the matching between those and the indexed documents.

3.1. Parsing

The first step in IR systems is about collecting the documents and provide them to the system in a "cleaned" and organized way. For this purpose, two classes have been developed in our system: the **WebDocument** and **WebDocumentParser** classes, which are responsible for reading the document and abstracting them in an organized Java object. We developed our parser class to read through the *.json* file format using *Jackson XML Parser*, and convert them into the specific Java class *WebDocument*.

The structure of the *.json* files is rather simple, and it is made as an array of anonymous elements with only two fields: *id* and *contents*, therefore the associated Java class will have the same structure. The most important thing about the parsing process is that it also provides a pre-processing step to clean the contents of the document. This step does not aim to perform detailed text analysis, as that responsibility is delegated to the analyzer component, therefore we just focus on the removal of HTML tags, of unwanted punctuation, of URLs and normalization of consecutive whitespaces.

All of this happens while accessing the list of documents in the *.json* file, so the process is performed only when actually accessing the required document when iterating the collection, rather than processing the whole collection at a single time. The **WebDocumentParser** provides access to the documents contained in a single file through the canonical iteration methods and returns the **WebDocument** object ready to be analyzed by the consumer **WebAnalyzer**.

3.2. Analyzing

Once we have access to the contents of our document, we must extract a sequence of textual elements - called *tokens* - to be provided to the indexer. These tokens represent the basic units upon which the search process operates. The main purpose of the analyzer component is to provide those tokens and apply some filtering and optimizations to improve the performance of our system.

To perform these operations, we developed our implementation of analyzer, called **WebAnalyzer**, which provides access to tokens through the **createComponents** method. The pipeline to generate a token is divided into several steps. First of all, we want to generate tokens from the text using one of the provided Lucene classes, like the **WhitespaceTokenizer**, the **LetterTokenizer** or the **StandardTokenizer**. Using one of these allows us to split the text into words or other components, which we can consequently analyze. After applying a **LowerCaseFilter** component to the token, to normalize the capitalization of the text, we want to remove the tokens which may not be relevant for our purpose: we use a **LengthFilter** to remove too short or too long tokens (setting thresholds in length to be in range 3-100), and finally a **StopFilter** component to eliminate the stopwords. In order to make the text more uniform, we also apply an **ElisionFilter** and **ASCIIFoldingFilter** components to remove elisions and regularize uncommon characters that may be present in the text, just before filtering the stopwords.

The last and probably most important step in our analysis pipeline is about the stemming process. We want our system to be able to match terms that might not be exactly written in the same way in the documents and queries, like (*e.g.* plural/singular nouns). Therefore, we apply a stemmer component at the end of the pipeline. Since the stemming process is a very delicate part of the analysis, we tried different configurations of the stemmer component, like the **Krovetz, SnowBall** and **Porter** stemmers, but also the Lucene's **FrenchLight/FrenchMinimal StemFilter**, since our collection is made of French queries and documents. At the end of the process, the WebAnalyzer generates a sequence of tokens ready to be consumed by our indexer.

To ensure flexibility and allow for easy experimentation with different analysis strategies, the WebAnalyzer component is configurable using an external XML file. This file specifies the parameters used in the analysis pipeline, such as the tokenizer type, minimum and maximum token length, the stemmer to be applied, and the activation of specific filters such as ASCIIFoldingFilter or ElisionFilter. The configuration is defined using an **AnalyzerParams** class, which is populated from the XML using Jackson annotations.

3.2.1. POS Tagging and Lemmatization

In order to improve the matching performance of our system, we tried to implement different versions of the analysis component. One of the first ideas was to take advantage of the OpenNLP [10] project to reduce the number of indexed terms and improve search results by expanding some types of words. In order to implement this kind of configuration, we tried to implement an **OpenNLPAnalyzer** class using OpenNLP models for sentence detection, tokenization, POS-tagging, and lemmatization. We also developed a POSTagFilter Tokenstream component to discard unwanted types of tokens. However, when trying to perform the analysis of the documents, we couldn't achieve a sufficiently fast execution time from the system, and therefore we dropped this implementation.

A second attempt at lemmatization was made using a more simple model based on Leff [11][12], a large-scale morphological and syntactic lexicon for French. In this case, we statically generated a copy of the collection documents containing the lemmatized terms, and then we used this collection as input to our system, but removing the stemming process from the analysis pipeline. In this case, the running time was very efficient, but we soon discovered that the lemmatization process didn't improve the matching performances of our system. Therefore, we dropped also this lemmatization process.

3.3. Indexing

Upon receiving the tokens generated from the analyzer's pipeline, the system must store the terms in the index. Lucene stores an inverted index, associating each term with a list of postings, that is the list of document ids containing that index term. In our configuration to improve the search capabilities of our system, we configured the index to also store the term frequencies and relative positions so that we could take advantage of the *tf-idf* metrics of each term and of phrase search. We also implemented a more complex indexing procedure, where each document is split into chunks, in order to improve the precision during the search phase, as better described in 3.3.1.

To manage the indexing process, we developed the **DirectoryIndexer** class, which allows us to parse and index an entire directory containing *.json* files, especially useful since in our case we are dealing with a sequence of "snapshot" directories, one for each month. Moreover, an efficient **MultiThreadIndexer** class has been developed, so that multiple *.json* files are indexed at the same time.

3.3.1. Chunk Indexing

In order to enhance the retrieval performance of our system, we also tried an alternative approach at index-time. Chunk indexing is a strategy designed to enhance retrieval granularity by dividing documents into semantically coherent text segments - called *chunks* - before indexing. The effectiveness of chunk-based approaches has been highlighted in both neural and classical IR frameworks, in particular, Yin and Schütze[13] operated on multiple levels of text granularity to improve matching performance across chunks. Their findings support the idea that representing and comparing textual content at different granular levels leads to improved relevance estimation.

Inspired by this principle, our system implements chunk indexing with overlapping windows of 10 sentences, as follows:

1. **Sentence-based Segmentation:** Documents are segmented into chunks using the `fr-sent.bin` French sentence model provided by OpenNLP[10]. Sentences are then grouped in chunks of 10 sentences each.
2. **Sliding Window with Overlap:** To preserve semantic continuity across chunks, a sliding window approach is used. Each new chunk begins after reusing the last m sentences (3 in this case) of the previous chunk as overlap.
3. **Unique Identification:** Each chunk is indexed as an independent Lucene document with a unique identifier formatted as `<docID>_<chunkIndex>`, ensuring traceability to its source document.

During retrieval, the system operates at the chunk level, but the results are aggregated at the document level using a simple yet effective post-processing step:

- For each original document, only the highest-scoring chunk is retained.
- The final score assigned to the document corresponds to the score of this best chunk, thus reflecting its most relevant passage.

This strategy enables both fine-grained relevance matching and efficient document-level ranking, particularly beneficial in multilingual or domain-diverse collections. Further considerations on the impact of chunking on retrieval performance are discussed in Section 5.8.

3.4. Searching

Moving on with the "online" section of our system, the *Searcher* class is the component tasked with interpreting user queries and scanning indexed documents to identify the most relevant matches. In order to perform the searching process, we also developed a *TopicParser* class, which is responsible to parse the queries file and pass them with their id to the searcher, in a similar way to what we have seen when parsing the *.json* documents. Several components have been tried to further improve the search phase, where the actual matching is performed through the **BM25** model.

3.4.1. BM25

The next step involves retrieving relevant documents based on the queries submitted. This process entails identifying the documents that match most closely to each query, using scoring functions to evaluate their relevance. Each document is assigned a score and the results are ranked accordingly, from highest to lowest, under the assumption that the highest scoring documents are more relevant to the query. Among these scoring methods, the BM25 ranking function, part of the "Best Match" (BM) family of retrieval models, stands out. Despite its simplicity, BM25 remains highly effective and continues to perform competitively against more modern retrieval techniques.

3.4.2. Queries

The queries provided by the LongEval team consist of a set of files, one for each monthly snapshot provided. It is possible to work with two different formats, *.trec* files or *.txt* files. We chose to work with *.txt* files for simplicity, but a parser was developed for each version in order to generate the corresponding **QualityQuery** object required by Lucene.

Query terms are then analyzed and used to generate more powerful queries, using different techniques to improve the search result. In the following subsections we describe our implemented procedures.

3.4.3. Fuzzy Search

We experimented with a fuzzy search extension after observing a high incidence of typographical errors in user queries, like *"amurerie"*, which should be *"armurerie"*. By replacing exact term matching with a Levenshtein-based fuzzy query, we aimed to recover relevant documents despite misspellings. However, contrary to our expectations, the fuzzy approach consistently underperformed compared to strict matching: the relaxation of edit distance constraints introduced substantial noise and lowered precision, ultimately degrading overall retrieval effectiveness. This surprising outcome led us to favor the original exact-matching pipeline for the final system configuration.

3.4.4. Pseudo-Relevance Feedback

Pseudo-Relevance Feedback (PRF) is an unsupervised technique used to improve retrieval effectiveness by automatically reformulating the query. Unlike traditional relevance feedback, it does not rely on human interaction. Instead, PRF expands the user's query by leveraging terms from top-ranked documents retrieved during an initial search. The method assumes that the most relevant terms can be extracted from the top-k results and used to enhance recall.

Our implementation follows these main steps:

1. **Initial Retrieval:** Execute the original query using BM25 to retrieve the top k documents.
2. **Term Extraction:** For each of the top documents, tokenize the text using the same analyzer as during indexing. Filter out stopwords and tokens with inappropriate length. Then, compute term frequency (TF) and document frequency (DF) per term.
3. **Scoring and Ranking:** Compute a BM25-inspired weight for each candidate expansion term:

$$\text{score}(t) = \text{TF}(t) \cdot \log\left(1 + \frac{N - \text{DF}(t) + 0.5}{\text{DF}(t) + 0.5}\right)$$

   where N is the total number of documents and $\text{DF}(t)$ is the document frequency of term t.
4. **Query Expansion:** Select the top-n scoring terms and construct a new query combining:
   - The original query, boosted by a weight α
   - The expansion terms, boosted by a lower weight β
5. **Final Retrieval:** Submit the expanded query to obtain the final re-ranked list.

Formally, the final query Q' is built as:

$$Q' = \alpha \cdot Q + \beta \cdot \sum_{t \in T_{\text{exp}}} t$$

where T_{exp} is the set of top expansion terms.

3.4.5. Proximity Search

Proximity search enriches traditional term-based retrieval by introducing a measure of how closely query terms co-occur within a document, which often signals a stronger semantic relationship. Documents in which the full set of query terms appears within this slop window receive a score boost, reflecting their contextual cohesion. For example, the query "red house brick" could be used to retrieve documents that contain phrases like "red house built with bricks", while in the meantime avoiding documents where the words are scattered or spread across. However, as previously mentioned, the documents must still contain all query terms to match the full proximity constraint. So to further capture cases where only subsets of the query are present, we automatically generate every pair and triplet of query terms and apply the same proximity criterion to each combination. We limit the subsets to pairs and triplets to avoid exceeding Lucene's BooleanQuery clause limit of 1024, which would otherwise be reached quickly with longer queries. These proximity constraints are added as optional clauses in the final query, ensuring that documents with locally clustered terms are boosted in rank. Later, we will discuss the tuning of the slop parameter and how its value can influence system performance.

3.4.6. Synonyms

In order to try to improve the performance, we implemented a query expansion technique with synonyms, exploiting the semantic lexicon **WOLF** (WordNet Libre du Français) [14]. This technique allows words semantically similar to the original term to be included in the query, increasing the probability of retrieving relevant documents even if expressed with different terms.

During the Lucene query construction, each token is analyzed and, if the expansion with synonyms is enabled, the *WolfManager* module is queried. This module manages a dictionary of synonyms extracted from the XML WOLF resource. For each token, if synonyms are present, a SynonymQuery is created that includes both the original term and its synonyms. This query is then added to the global Lucene query using the BooleanQuery builder, with the SHOULD operator, to expand the coverage without penalizing the results based on the original term.

3.4.7. Reranking

To improve the quality of the results, the first k retrieved documents are subjected to a semantic re-ranking phase. The value of k will be discussed in Section 5.9. For each document, the textual content and the query are sent to a **local server**, which hosts a SentenceTransformer model: `all-roberta-large-v1` [15], available on Hugging Face [16]. The model is built upon the RoBERTa-large architecture, a robustly optimized variant of *BERT*, and is fine-tuned on over one billion sentence pairs to produce semantically meaningful embeddings. Given an input, the model maps it into a dense 1024-dimensional vector space such that semantically similar sentences are located close together. This makes it well-suited for tasks like reranking, semantic search, and clustering.

The server computes these steps:

- It employs the model, which performs a vector representation (embedding) of both the query and the document and evaluates their similarity through the scalar product.
- The score obtained, between 0 and 1, reflects the semantic relevance between the two texts so it's **weighted** and **added** to the original Lucene score, in order to obtain a new ranking value.
- The documents are then reordered based on this aggregated score.

This approach allows us to combine the efficiency of traditional retrieval with the generalization and semantic understanding capabilities offered by Deep Learning models.

3.5. Running the system

Once our components have been developed, we implemented a main **SystemRunner** class to coordinate the entire pipeline execution. The indexing and searching parts can be executed at different times, to simulate the *offline* and *online* deployment phases. The parameters of the system are passed to the components through constructors and *XML* configuration files, in order to improve the flexibility of the system.

The final complete overview of the system is shown in Figure 1.

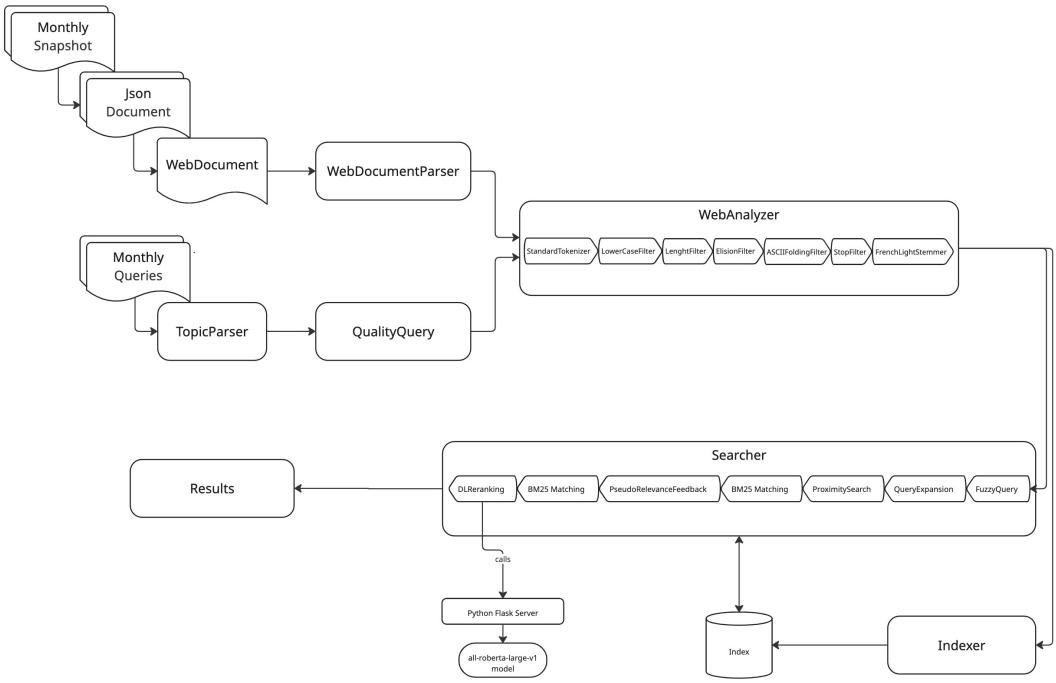


Figure 1: General overview of a possible configuration of our implemented system. We can see how the analyzer is a key element used both in the offline and online phases, to analyze documents and queries. The searcher uses a python flask server to retrieve the score for each document using a LLM model and rearranges the results accordingly. The chunk indexing process is not highlighted here.

4. Experimental Setup

As already discussed, the system has been developed in Java through the Apache Lucene Library, but a few side components have been developed in Python. The source code can be found at https://bitbucket.org/upd-dei-stud-prj/seupd2425-3ds2a/src/master/, and as an evaluation measure, we used the metrics provided by the `trec_eval 9.0.7` tool, available at the https://trec.nist.gov/trec_eval/ repository.

The system has been run on the collection provided by the LongEval team at https://clef-longeval.github.io/data/. We used our personal computers to test our system, where the most expensive runs have been completed using the hardware described in Table 1.

Table 1
Most powerful hardware available

	CPU	RAM	GPU
System 1	i7-10750H	16GB	GTX 1660-Ti
System 2	Ryzen 9 3900x	32GB	RTX 3070

4.1. Training data and Evaluation

The training collection is made up of several components. LongEval team has given us a timeline sequence of documents, extracted from the Qwant search engine through a sequence of months, from June 2022 to February 2023, with every month in a specific subfolder. For every monthly snapshot, the corresponding list of queries to submit has been provided at https://github.com/clef-longeval/clef-longeval.github.io/tree/master/collection. Moreover, to understand how the system performed each month, a corresponding *qrels* file has been provided for each month.

The evaluation metrics have been computed using the *-m all_trec* parameter of the *trec_eval* executable, which will give us several metrics to examine. For training data, we mostly focused on the **nDCG** and **MAP** metrics, in order to improve the overall result of the system.

5. Results and Discussion

The first thing we need to do to improve our system is to set a reference score for our metrics. In order to do that, we configured our system to use only the main components of Lucene. Upon establishing a baseline evaluation score, we could then try to develop new strategies to improve the final result of the system.

5.1. Baseline performance using Lucene library

To set a baseline for our next runs, we used a simple configuration of the *Indexer, Analyzer,* and *Searcher* components, as described in Table 2.

Table 2
Baseline system's configuration. The indexing process doesn't use the chunk-based approach.

	Used components from Lucene
Tokenizer	StandardTokenizer
Text normalization	LowercaseFilter, ElisionFilter, ASCIIFoldingFilter
Filtering	LengthFilter(3–100) and StopFilter (French and English)
Stemming	FrenchLightStemmer
Matching model	BM25

When running this base system on the provided training dataset, we achieved very different results depending on the considered month. We repeated the measurements several times and came to the conclusion that some months have very difficult topics or strange judgments, but probably also because some documents have different languages and may not be processed in an efficient way. Nevertheless, we achieve a complete overview of the baseline system's performance in the Table 3.

Table 3
Baseline system's performance. Notice how the score rises from June to January.

Month	MAP	nDCG
2022-06	0.1174	0.2076
2022-07	0.1208	0.2089
2022-08	0.1325	0.2168
2022-09	0.1228	0.2062
2022-10	0.2078	0.3104
2022-11	0.2122	0.3136
2022-12	0.2192	0.3244
2023-01	**0.2232**	**0.3277**
2023-02	0.2019	0.2969

5.2. Stemming against Lemmatizing

Even if we couldn't manage to achieve an efficient system using OpenNLP for lemmatizing the documents, we tried a simpler way to generate a lemmatized version of the collection. Using a dictionary-based lemmatizer [11], we generated a lemmatized copy of the collection, including the queries, and fed them to our system, taking care of disabling any active stemmer. The results are depicted in Figure 2 and Figure 3. The comparison with the baseline system highlights how the performance is pretty much aligned with the stemmed version of the collection, but it is important to say that the lemmatizer used is probably too simple. Nowadays our systems could take more advantage of LLM models, and from a better lemmatization it would be possible to further expand each term to improve the final result of the system.

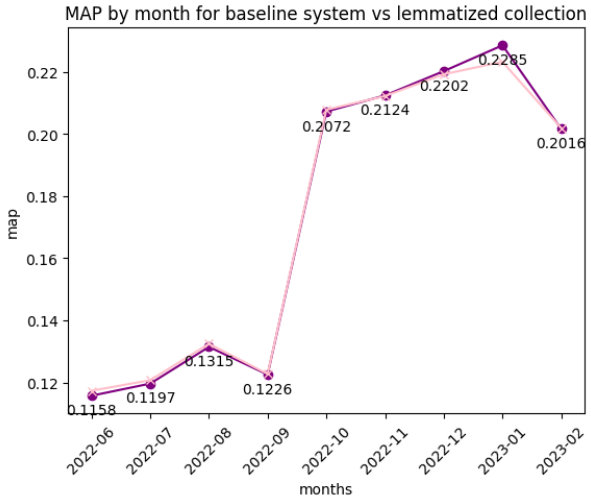


Figure 2: MAP score of the baseline system over the original collection against the lemmatized version of the collection. Reported numbers are the scores obtained using lemmatized collection.

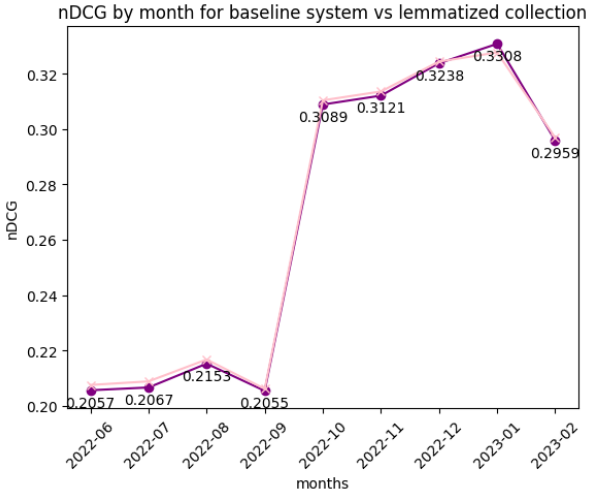


Figure 3: nDCG score of baseline system over the original collection against the lemmatized version of the collection. Reported numbers are the scores obtained using lemmatized collection.

5.3. Choice of stemmer

To assess the effectiveness of different text analysis strategies, we performed a series of experiments combining various tokenizers (Standard, Letter, and Whitespace) with multiple stemmers (FrenchLight,

FrenchMinimal, Snowball, and None). All configurations were tested on the 2022-06 dataset.

The evaluation focused on two main retrieval metrics: Mean Average Precision (MAP) and nDCG. Across the board, the FrenchLight stemmer consistently delivered the best results, particularly when paired with the StandardTokenizer, achieving the highest MAP (0.1174) and competitive nDCG scores. Configurations using the LetterTokenizer also performed well, providing strong baseline alternatives.

Interestingly, the use of stopword filtering did not lead to performance improvements. This trend was consistent across both French and English stoplists, suggesting that aggressive stopword removal may be useless in this context.

Despite the fact that stopword filtering did not improve retrieval performance in our current experimental, this feature will remain part of the default analysis pipeline. There are several reasons behind this decision. First, stopword removal improves robustness by eliminating high-frequency, low-content terms that might introduce noises. Second, stopwords can significantly affect index size and efficiency, especially when dealing with large corpora. Lastly, including stopword filtering maintains compatibility with standard IR practices and facilitates future integration of more advanced techniques, such as query expansion or relevance feedback, which often employ a cleaner input signal. Some interesting results can be seen in Table 4.

Table 4
Performances for various tokenizer and stemmer configurations on the 2022-06 dataset.

Configuration	MAP	nDCG
Standard+FrLight	**0.1174**	0.2076
Standard+FrMin	0.1159	0.2057
Standard+Snow	0.1157	0.2060
Standard+Porter	0.1153	0.2054
Standard+Krow	0.1140	0.2045
Letter+FrLight	0.1167	**0.2078**
Whitespace+FrLight	0.1093	0.1942
No stopwords	**0.1175**	0.2076
Baseline	0.1174	0.2076

5.4. BM25 Parameters Tuning

To evaluate the impact of the parameters of the ranking function on retrieval performance, we experimented with different configurations of the BM25 similarity function, which is the default scoring method used in many modern IR systems.

We evaluated multiple parameter settings of BM25, particularly focusing on variations of the parameters k_1 (term frequency saturation) and b (document length normalization). The baseline run used the default Lucene settings, while alternative runs explored more aggressive and conservative configurations.

Among the settings tested, the default setting (with parameters k_1 = 1.2 and b = 0.75) achieved the best overall performance. This setup outperformed alternative configurations, including more aggressive ones such as BM25(2.0, 1.0), which, although slightly better in recall (R@1000 = 0.4113), showed less consistent results in early precision and overall MAP. In contrast, the more conservative configuration BM25(0.9, 0.4) significantly underperformed in both early and deep precision. An additional run with BM25(0.5, 0.0), which removed length normalization entirely, was tested but, as expected, showed the worst scores across all metrics.

These results suggest that the default BM25 function offers the best trade-off between precision and robustness for our dataset, and thus was selected as the standard configuration in our system.

Table 5
Comparison of BM25 parameter settings on the 2022-06 dataset.

k_1, b	MAP	nDCG	nDCG@10	R@1000	Notes
(1.2, 0.75)	**0.1203**	**0.1884**	**0.1456**	**0.4058**	**Default (best overall)**
(0.9, 0.4)	0.1176	0.1844	0.1421	0.3963	Conservative setting
(2.0, 1.0)	0.1170	0.1863	0.1404	0.4113	Higher recall, more noise
(0.5, 0.0)	0.1101	0.1750	0.1339	0.3799	No normalization (worst)

5.5. Synonyms

In this section, we are going to talk about the results achieved through synonyms query expansion. The experiment setup was the one used in the baseline. The synonyms included in the queries are analyzed with the same analyzer used for indexing the documents and, furthermore, are assigned a weight of 0.5 to reduce their significance compared to the words originally present in the queries.

Table 6
Performance of the system using synonyms against performance of the baseline system. Notice how the scores are very similar, but the effort of going through query expansion with synonyms is not worth the effort.

Month	Synonyms		Baseline	
	MAP	nDCG	MAP	nDCG
2022-06	0.1150	0.2043	0.1174	0.2076
2022-07	0.1188	0.2055	0.1208	0.2089
2022-08	0.1302	0.2135	0.1325	0.2168
2022-09	0.1206	0.2034	0.1228	0.2062
2022-10	0.2026	0.3050	0.2078	0.3104
2022-11	0.2080	0.3088	0.2122	0.3136
2022-12	0.2160	0.3201	0.2192	0.3244
2023-01	**0.2200**	**0.3234**	0.2232	0.3277
2023-02	0.2000	0.2935	0.2019	0.2969

As shown in Table 6, query expansion with synonyms did not lead to any performance improvements. In all months analyzed, the MAP and nDCG scores are slightly lower than those of the baseline. This suggests that including synonyms, even with a reduced weight, tends to introduce noise rather than add value to the queries. Based on these results, we decided not to adopt this strategy in future configurations.

5.6. Proximity Search

Here we present the results obtained through the implementation of proximity search. As anticipated before, the tuning of the slop parameter was a key point for obtaining the best results. Several test runs were conducted to determine the optimal slop setting, and the results showed that a value of 50 consistently yielded the best performance. Therefore, this value was selected for the final configuration. A possible explanation for this result is that smaller slop values may be too restrictive, missing relevant documents where query terms appear slightly farther apart. On the other hand, larger values might introduce too much noise, retrieving documents with weaker term associations. A slop of 50 appears to strike the right balance between flexibility and precision, allowing for meaningful proximity without overly relaxing the positional constraints. As discussed in subsubsection 3.4.5, we explored two approaches: the first uses a *PhraseQuery* to match documents where all query terms appear in the same order as in the original query; the second extends this by incorporating additional proximity constraints based on all possible pairs and triplets of query terms, regardless of their order, using *SpanNearQuery* with `inOrder=false`. As previously discussed, in the unordered case, we restricted the proximity constraints to all possible pairs and triplets of query terms, rather than the full query, in order to avoid exceeding the BooleanQuery clause limit.

Table 7 compares the system performance when using only the full-query proximity constraints (*PhraseQuery*) against the extended approach that also includes unordered pairs and triplets (*PhraseQuery + SpanNearQuery*). Firstly, it can be observed that the implementation of proximity search leads to a consistent and substantial improvement in both MAP and nDCG scores compared to the baseline. In particular, the maximum change is observed in January with an increase of 7.2% for the MAP and 4.15% for the nDCG. Regarding the comparative performance of the two strategies, a trade-off emerges: relative to the PhraseQuery-only approach, the extended method involving unordered pairs and triplets leads to a slight decrease in MAP, while consistently improving nDCG (except for January). This suggests that incorporating unordered term combinations may retrieve more relevant documents overall, even if precision at the top ranks is marginally affected.

Table 7
Comparison between the baseline system and the two implementations of PhraseQuery, one with also the SpanNearQuery technique.

Month	PhraseQuery		PhraseQuery + SpanNearQuery		Baseline	
	MAP	nDCG	MAP	nDCG	MAP	nDCG
2022-06	0.1267	0.2160	0.1276	0.2189	0.1174	0.2076
2022-07	0.1294	0.2167	0.1293	0.2185	0.1208	0.2089
2022-08	0.1398	0.2237	0.1384	0.2244	0.1325	0.2168
2022-09	0.1304	0.2128	0.1311	0.2152	0.1228	0.2062
2022-10	0.2219	0.3222	0.2201	0.3227	0.2078	0.3104
2022-11	0.2240	0.3236	0.2226	0.3241	0.2122	0.3136
2022-12	0.2311	0.3342	0.2300	0.3349	0.2192	0.3244
2023-01	**0.2393**	**0.3413**	**0.2367**	**0.3407**	0.2232	0.3277
2023-02	0.2164	0.3094	0.2141	0.3099	0.2019	0.2969

5.7. Pseudo Relevance Feedback

We tested two different configurations of Pseudo Relevance Feedback (PRF), with the aim of evaluating the impact of different parameter settings. The first configuration, referred to as **PRF1**, is more conservative and uses the following settings: `"topDocsForPRF": 10`, `"topTermsToAdd": 3`, `"originalBoost": 1.0`, and `"expansionBoost": 0.3`. The second configuration, **PRF2**, is more aggressive and relies on `"topDocsForPRF": 20`, `"topTermsToAdd": 5`, `"originalBoost": 0.8`, and `"expansionBoost": 0.5`.

It is worth noting that PRF is typically more effective in scenarios where initial precision is high (e.g., high P@5 or P@10), as it assumes that top-ranked documents are relevant and can guide useful expansion. Therefore, evaluating PRF in isolation on a general baseline setup may not fully reflect its potential. Nevertheless, this comparison provides useful insights for tuning the PRF strategy.

As expected, both PRF configurations resulted in a decrease in nDCG when applied on top of the baseline system. However, the comparison between PRF1 and PRF2 helps highlight the relative benefits of a more conservative versus a more aggressive expansion approach.

Later in this report, we will discuss how PRF interacts with other retrieval enhancements and whether its impact changes when combined with more effective base configurations.

5.8. Chunk Indexing

To investigate whether finer-grained indexing could enhance retrieval effectiveness, we experimented with a chunk indexing approach. The documents were split into overlapping chunks of 10 sentences each, with a shared window of 3 sentences.

As shown in Figure 5 and Figure 6, chunk indexing consistently slightly underperforms compared to the baseline. Both MAP and nDCG values are lower throughout the time period, suggesting that

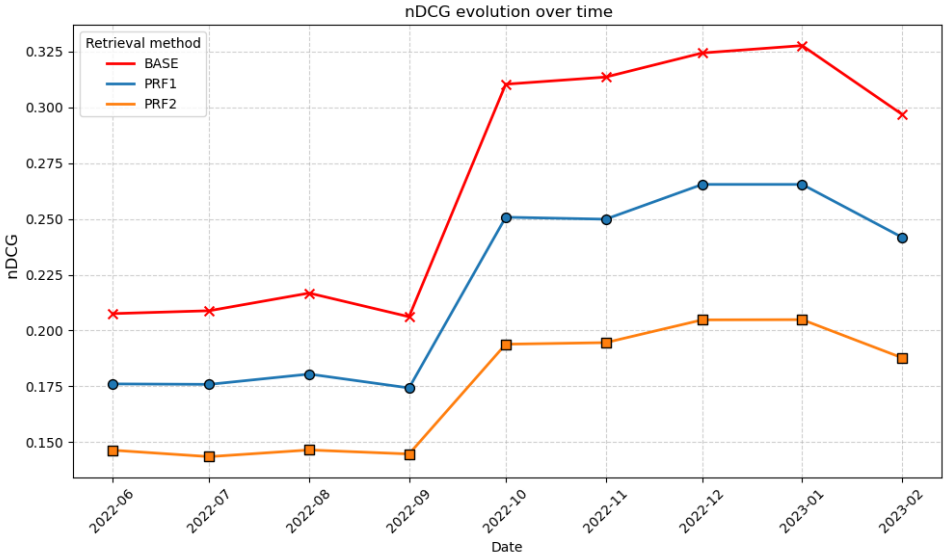


Figure 4: nDCG evolution over time for baseline, PRF1 and PRF2 configurations.

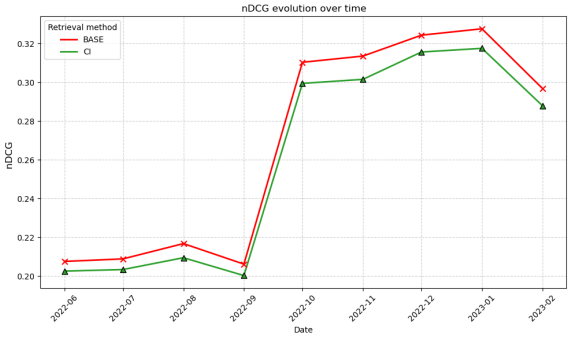


Figure 5: Chunk Indexing nDCG evolution over time

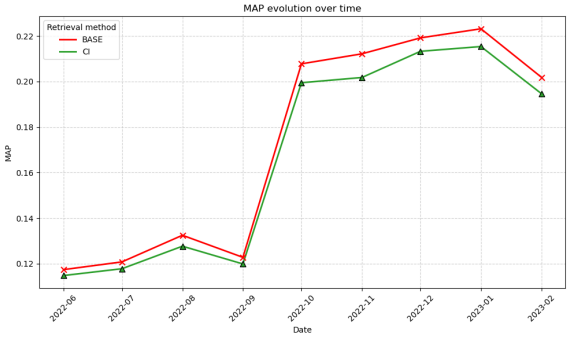


Figure 6: Chunk Indexing MAP evolution over time

splitting documents into sentence-based overlapping chunks, in this configuration, degrades retrieval performance.

In this scenario, chunking probably introduces noise and dilutes term co-occurrence signals, making it harder for the system to correctly prioritize relevant documents.

5.9. Reranker

As discussed in subsubsection 3.4.7, we need to determine a threshold for the number of documents to be reranked. Based on the work by Basaglia et al. [4], we observed that reranking the top 150 documents provided a good balance between efficiency and effectiveness, so we selected this value. Below is a table showing the performance achieved by our system each month, using reranking and proximity search (approach with *PhraseQuery + SpanNearQuery*), which are techniques that improve our system.

As shown, reranking leads to improvements in both MAP and nDCG scores over proximity search (*PhraseQuery + SpanNearQuery* approach) results across all months. The greatest gains are observed in November, with MAP increasing by 5.1% and nDCG by 2.7%.

5.10. Combined Strategies

In this section, we evaluate the performance of various combined approaches, all built on top of the Proximity Search method. Given the consistently strong results achieved by PS alone, it serves as the

Table 8
System performance with reranking and proximity search (*PhraseQuery-SpanNearQuery* approach.)

Month	Reranking + Proximity		Baseline	
	MAP	nDCG	MAP	nDCG
2022-06	0.1338	0.2255	0.1174	0.2076
2022-07	0.1351	0.2243	0.1208	0.2089
2022-08	0.1428	0.2291	0.1325	0.2168
2022-09	0.1367	0.2208	0.1228	0.2062
2022-10	0.2288	0.3304	0.2078	0.3104
2022-11	0.2339	0.3328	0.2122	0.3136
2022-12	0.2356	0.3403	0.2192	0.3244
2023-01	**0.2436**	**0.3472**	0.2232	0.3277
2023-02	0.2225	0.3174	0.2019	0.2969

foundation for all hybrid strategies evaluated here.

We consider combinations such as PS with Chunk Indexing (PS_CI), PS with Pseudo Relevance Feedback (PS_PRF1), and the full combination PS_CI_PRF1. The goal was to assess whether the integration of multiple retrieval enhancements could provide cumulative improvements.

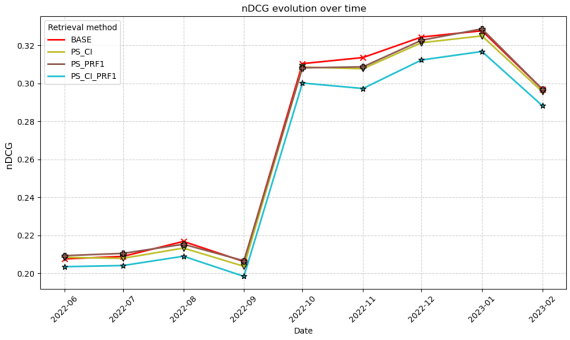


Figure 7: nDCG: baseline vs. hybrid strategies

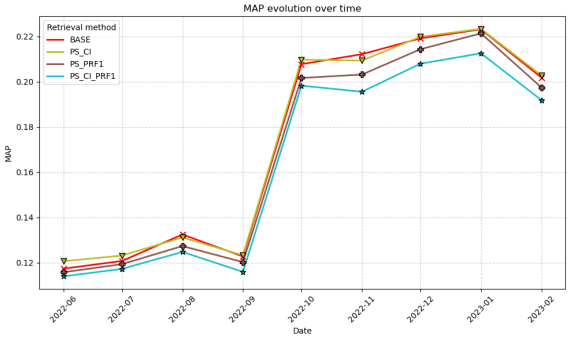


Figure 8: MAP: baseline vs. hybrid strategies

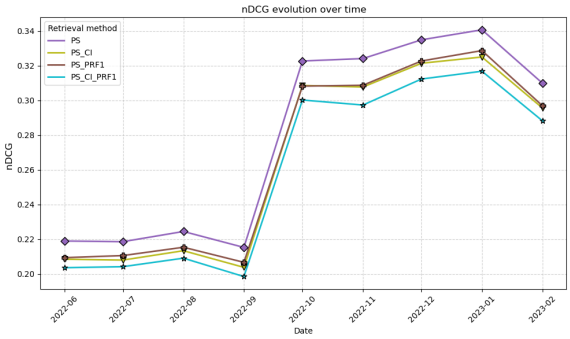


Figure 9: nDCG: PS vs. hybrid strategies

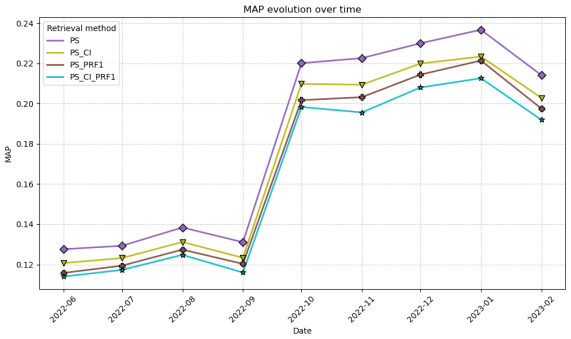


Figure 10: MAP: PS vs. hybrid strategies

As the figures above show, none of the combined strategies significantly outperforms the standalone PS method. In some cases, such as PS_CI_PRF1, the addition of other techniques even slightly degrades performance. This is particularly evident in the MAP plots, where more complex configurations tend to underperform compared to PS alone.

One possible explanation is that Pseudo Relevance Feedback (PRF1) may not yet be effective in this context, possibly due to insufficient initial precision. PS alone may not reach high enough early precision to make expansion terms reliably informative. Alternatively, the PRF configuration may

require different fine-tuning when applied on top of PS.

Despite this, the results demonstrate the robustness of the PS method. Even when combined with less effective strategies, PS manages to retain a high level of performance across time. This suggests that PS, as implemented, remains the most reliable and effective enhancement among those tested in this study.

5.11. Training results

In this section, we compare the best-performing configurations across various implemented techniques. These configurations are the runs we have selected for submission to the LongEval Conference:

- The best system that doesn't use reranking.
- The best overall system.
- A system using only chunk indexing.
- A system using pseudo-relevance feedback and proximity search.
- A system using proximity search, chunk indexing and pseudo-relevance feedback.

Each of the systems will use the configuration of the baseline, that we recall being the one reported in Table 2. The proximity search approach is the one considering also pairs and triples (*PhraseQuery-SpanNearQuery*)

Table 9 presents, for each system configuration, the best result achieved across all months, which corresponds to the score of January.

Table 9
Performance of the submitted configurations on the training dataset

Label	System	MAP	nDCG
System 1	Baseline - Pseudo relevance feedback 1 - Proximity search	0.2214	0.3287
System 2	Baseline - Chunk indexing	0.2154	0.3176
System 3	Baseline - Proximity search (best performing system with no reranking)	0.2367	0.3407
System 4	Baseline - Pseudo relevance feedback 1 - Chunk indexing - Proximity search	0.2126	0.3168
System 5	Baseline - Proximity search-Reranking (best performing system)	**0.2436**	**0.3472**

From the results shown in Table 9, we can draw several key insights. First, proximity search alone (System 3) already provides a significant improvement over the baseline, even without reranking. However, the best overall performance is achieved by combining proximity search with reranking (System 5), which outperforms all other configurations in both MAP and nDCG, highlighting the importance of reranking in enhancing retrieval quality. Interestingly, the combination of proximity search, chunk indexing, and pseudo-relevance feedback (System 4) does not surpass the performance of simpler approaches, suggesting that the integration of multiple techniques does not necessarily lead to additive gains. Chunk indexing alone (System 2) shows a loss in performance, especially in terms of nDCG. These findings support the submission of System 3 as the best non-reranking system and System 5 as the most effective overall configuration.

The interpolated Precision-Recall curves shown in Figure 11 confirm the results observed in the tabular metrics. System 5 consistently demonstrates the highest precision across nearly all levels of recall, reinforcing its position as the overall best-performing configuration, closely followed by System 3. In contrast, System 4 clearly underperforms relative to the other systems, suggesting that combining multiple techniques—as done in this configuration—may introduce additional complexity without yielding significant performance gains. Notably, Systems 3 and 5 exhibit remarkably similar trends, with System 3 performing slightly worse; this may be attributed to their shared reliance on proximity search technique. Another noteworthy observation is that, within the recall interval approximately between 0.55 and 0.75, Systems 1, 2, and 4 outperform the proximity search-based systems.

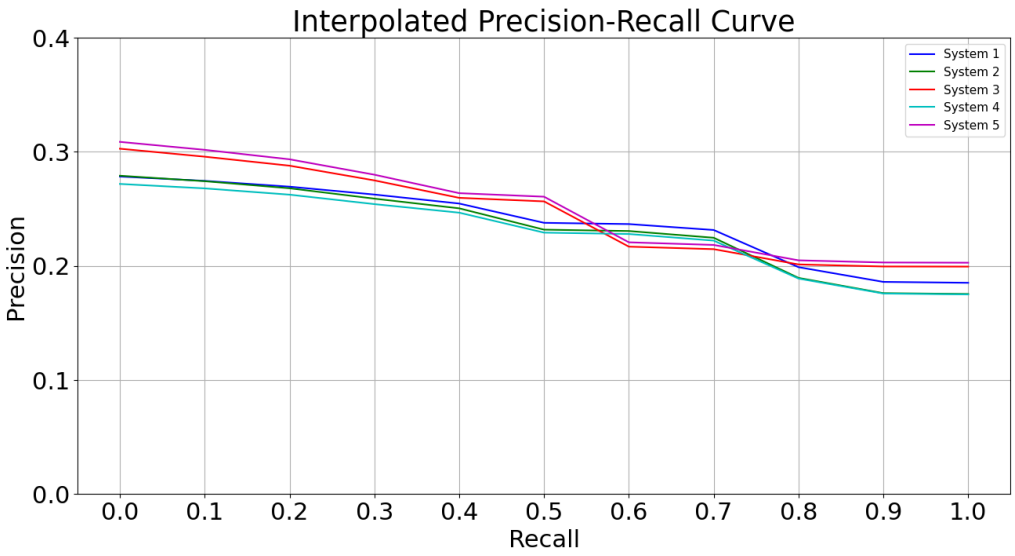


Figure 11: Interpolated Precision Recall curves on the training dataset

5.12. Training results on updated training dataset

We conducted experiments also on the updated version of the training queries. Since all previously evaluated systems yielded the same qualitative conclusions despite improvements in their metric values, we have omitted their full result tables to avoid verbosity and redundancy. Instead, we report only the proximity search technique that yielded qualitatively different results compared to the first version training set.

Table 10
Comparison between the baseline system and the two implementations of proximity search on the new version of the training set.

Month	PhraseQuery		PhraseQuery + SpanNearQuery		Baseline	
	MAP	nDCG	MAP	nDCG	MAP	nDCG
2022-06	0.2186	0.3260	0.2173	0.3259	0.2009	0.3097
2022-07	0.2113	0.3149	0.2097	0.3143	0.1971	0.3018
2022-08	0.2124	0.3132	0.2108	0.3127	0.1977	0.3003
2022-09	0.1955	0.2888	0.1961	0.2901	0.1828	0.2774
2022-10	0.2928	0.4025	0.2896	0.4005	0.2723	0.3847
2022-11	0.2929	0.4023	0.2893	0.3997	0.2718	0.3839
2022-12	0.2994	0.4119	0.2962	0.4097	0.2796	0.3947
2023-01	**0.3073**	**0.4173**	**0.3040**	**0.4150**	**0.2844**	**0.3977**
2023-02	0.3079	0.4115	0.3042	0.4093	0.2848	0.3913

Table 10 reveals a notable shift from our initial findings (Table 7): the variant with only *PhraseQuery* now outperforms the combined *PhraseQuery + SpanNearQuery* approach. Consequently, we will hereafter employ proximity search exclusively with *PhraseQuery*. Furthermore, by comparing results obtained using the updated training set (Table 10) with the first training set's results (Table 7) we can also notice a general boost of performances in all months. The performance trend of our system is consistent with the original training set obtaining also with this updated version the best performances in January.

5.13. Test results

CLEF also released a test set consisting of documents spanning from March 2023 to August 2023, again organized on a monthly basis, each accompanied by its corresponding set of qrels. To evaluate the effectiveness of the submitted systems, we computed the nDCG and MAP metrics on the test collections. In order to avoid overly verbose and redundant reporting, we selected a representative subset of months, each capturing a different stage in the temporal evolution of the test set. This strategy allows for a clearer analysis of how each IR system performs and adapts over time—an essential aspect of the LongEval task, which aims to identify systems capable of maintaining stable performance in the face of temporal drift.

The analysis was carried out for each of the five submitted systems across three temporal segments: the short-term test collection (March 2023), the mid-term collection (June 2023), and the long-term collection (August 2023). Table 11 presents the performance of the systems among March, June, and August.

Table 11
Comparison among short-term, mid-term and long-term collections.

Label	System	nDCG			MAP		
		March	June	August	March	June	August
System 0	Baseline approach	0.4348	0.4678	0.3590	0.3176	0.3521	0.2664
System 1	Baseline - Pseudo relevance feedback 1 - Proximity search	0.4120	0.4410	0.3424	0.2842	0.3166	0.2445
System 2	Baseline - Chunk indexing	0.4245	0.4544	0.3494	0.3084	0.3399	0.2584
System 3	Baseline - Proximity search (best performing system with no reranking)	0.4609	0.4860	0.3773	0.3482	0.3738	0.2878
System 4	Baseline - PRF1 - Chunk indexing - Proximity search	0.4208	0.4477	0.3455	0.2997	0.3290	0.2515
System 5	Baseline - Proximity search - Reranking (best performing system)	0.4700	**0.4950**	0.3819	0.3570	**0.3832**	0.2923

The table also reports the performance of the presented baseline approach on the test dataset. As we can see not every developed system improves the scores across months: Systems 1, 2, and 4 have a drop of 2%-5% in the nDCG score, and up to 10% in the MAP score. On the other side, the best-performing systems (Systems 3 and 4) show some relevant improvements: the System 3 increases the performance of the nDCG metric from 3% to 6%, and from 6% to 9% for the MAP, while the System 5 increases the performances in nDCG scores from 5% to 8% and in MAP score from 8% to 9%, and thus proving the great effectiveness in the use of reranking systems and query manipulation techniques, with respect to a more traditional system. Moreover the analysis of nDCG and MAP scores reveals a clear pattern: all five systems improve from March to a peak in June 2023, but then undergo a steady performance decline by August 2023. Between March and August, nDCG drops by approximately 17% – 19% and MAP by 14% – 18% across systems, indicating susceptibility to temporal drift. Notably, System 5—despite maintaining the largest lead—exhibits the greatest sensitivity to drift (-18.7% nDCG), whereas the worst system, System 1, shows the smallest decline (-16.9% nDCG). Another interesting fact is that the spread between the best and worst systems shrinks from 5.8 points in March to 3.95 points in August, indicating a sort of convergence of system efficacy under temporal drift.

From March to June, all systems register a clear uplift, but the magnitude varies. Systems 1 and 2 lead the pack with nDCG increases of approximately 7.0% each (MAP gains of 11.4% and 10.2%, respectively), whereas the more complex Systems 3 and 5 improve by only 5.5% and 5.3% in nDCG (MAP gains of 7.3% and 7.4%). This suggests that simpler indexing strategies may adapt more quickly to fresh data.

Overall, these findings suggest that although reranking and proximity search enhancements yield the strongest results, they remain vulnerable to evolving document distributions, underscoring the need for IR models that are more robust to temporal change.

In addition, statistical hypothesis tests were performed to assess whether the performance differences among the systems are statistically significant. This analysis is crucial to determine whether certain configurations genuinely contribute to performance improvements or if the observed gains are merely the result of test variance. In the following section, we provide a detailed explanation of the SHT methodology and present the results for both the evaluation metrics previously discussed.

6. Statistical Hypothesis Testing

Statistical Hypothesis Testing (SHT) provides a mathematical framework to draw statistical inferences from data. It involves comparing a null hypothesis H_0 against an alternative hypothesis H_1. The result is considered statistically significant if the observed data are unlikely to have occurred under the null hypothesis, based on a predefined threshold α, known as the significance level. If this condition is met, the null hypothesis is rejected; otherwise, we fail to reject it.

In this study, we apply Two-Way *ANalysis Of VAriance* (*ANOVA2*) as a statistical test: it examines the influence of two different variables, which in our case are the systems and the topics used. ANOVA2 is used to evaluate the difference between the means of more than two groups, which fits our necessity to compare five different IR systems. In ANOVA2, the hypotheses are as follows:

- H_0 - the means of all groups are equal;
- H_1 - at least 2 groups have different means.

Whenever the ANOVA test reveals statistically significant differences (i.e., H_0 is rejected), we further conduct a *Tukey's Honestly Significant Difference* (*HSD*) test as a post-hoc analysis. This allows pairwise comparisons between group means to identify which differences are statistically meaningful. For all tests we adopt a significance level $\alpha = 0.05$.
In this section we report the analysis conducted on the test collection.

6.1. Short-term

To visually complement the numerical findings and support the subsequent statistical testing, in Figure 12 we report the box plots of the Average Precision (AP) and nDCG scores for each system on the short-term collection. These graphical representations provide insights into the distribution, variability, and central tendencies of system performance.

As observed in the AP box plot, all systems exhibit very similar distributions, with nearly identical interquartile ranges and whiskers, suggesting comparable variability. In all systems, the mean (green dotted line) is higher than the median (orange line), indicating slight right-skewness—where a few high-performing queries raise the average. All systems reach the maximum AP value of 1.0 and a minimum close to 0.0, revealing the presence of both very successful and very weak queries. The absence of outliers further suggests consistent performance across queries.

Similarly, the nDCG box plot confirms these trends: the five systems nearly overlapping distributions with consistent whisker lengths and interquartile ranges. The means exceed the medians for all systems, indicating a right-skewed distribution here as well.

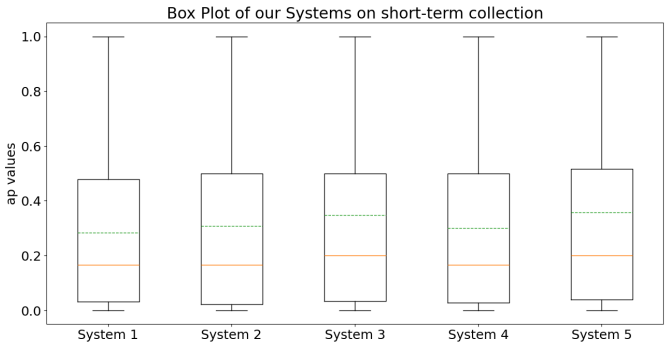
These findings highlight that, despite minor differences, the systems behave similarly on the short-term collection—reinforcing the need for formal hypothesis testing to establish whether observed differences are statistically significant.

Table 12 and Table 13 report the results of the ANOVA2 test conducted on the short-term collection for MAP and nDCG, respectively.

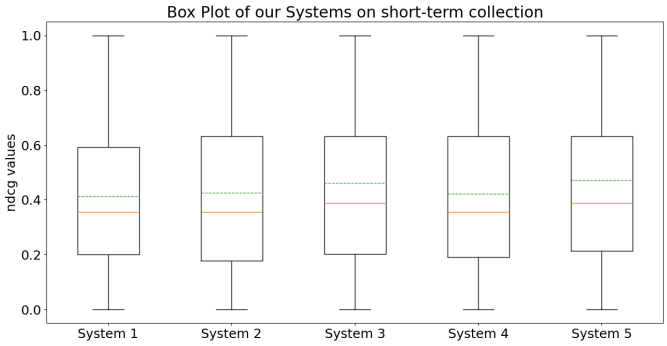
The sum of squares (SS) column quantifies the total variation attributed to each source—either between systems (Columns), between topics (Rows), or residual error. The degrees of freedom (df) indicates how many independent components contribute to each source's variability.

The mean square (MS) is obtained by dividing SS by the corresponding degrees of freedom and reflects the variance contribution of each source. The F column reports the F-statistic, which tests whether the observed variability across systems or topics is significantly greater than what would be expected by chance. Finally, the Prob>F column shows the p-value associated with each F-statistic.

In both tests, the p-value for the Columns source is well below the 0.05 significance threshold, providing overwhelming evidence that the five systems do not perform equally. The Rows component, which reflects topic variability, is also highly significant, indicating that the choice of topic greatly impacts system performance. This suggests that performance varies not only across systems but also



(a) AP values



(b) nDCG values

Figure 12: Box plots of the five systems evaluated on the short-term collection using AP and nDCG.

Table 12
ANOVA2 AP on short-term collection

Source	SS	df	MS	F	Prob>F
Columns	20.2759	4	5.0689	232.7448	9.60e-196
Rows	2639.2223	5082	0.5193	23.8451	0
Error	442.7277	20328	0.0217		
Total	3102.2261	25414			

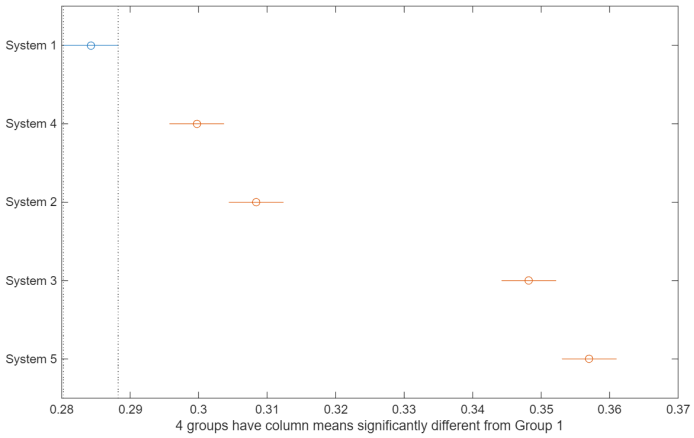
Table 13
ANOVA2 nDCG on short-term collection

Source	SS	df	MS	F	Prob>F
Columns	13.7145	4	3.4286	243.7508	7.41e-205
Rows	2189.6206	5082	0.4308	30.6308	0
Error	285.9366	20328	0.0140		
Total	2489.2718	25414			

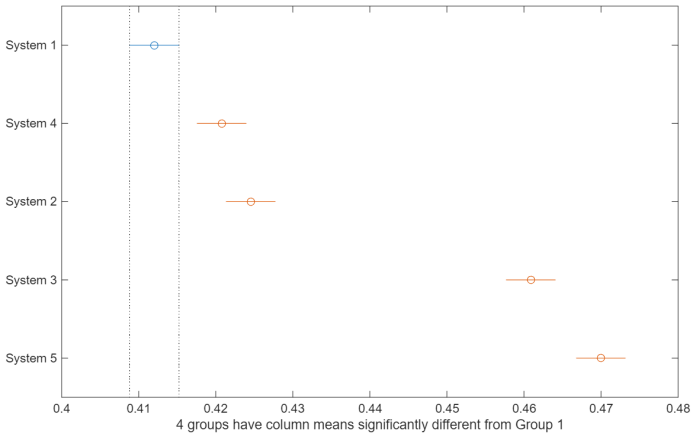
across queries, underscoring the importance of robust performance across diverse topics. Thus, we reject the null hypothesis H_0.

After discovering that the 5 systems are different, it is useful to compare systems pairwise in order to understand where the difference comes from. For this purpose, as said before, we will employ *Tukey's Honestly Significant Difference* test, which is used in order to complement the statistical results by illustrating the pairwise comparisons between systems, highlighting which differences in mean performance are statistically significant. Figure 13 shows a comparison among the mean of the different groups. System 1 has been selected as the reference group, as indicated by the vertical dotted line

corresponding to its mean value. The plots show the mean performance of each system along with their confidence intervals, both for AP and nDCG. We can see that four out of the five systems exhibit statistically significant differences in their mean values when compared to System 1, both for AP and nDCG. Specifically, the confidence intervals for Systems 2, 3, 4 and 5 do not overlap with that of System 1, confirming that the differences are not due to chance at the selected significance level $\alpha = 0.05$. The output of the test is reported in Table 14 and Table 15 for AP and nDCG, respectively. The p-values lower than 0.05 are shown in bold, meaning that we reject the null hypothesis H_0.



(a) AP values



(b) nDCG values

Figure 13: Plot of the differences in means across multiple groups, illustrating the variation within and between groups in the short-term collection.

6.2. Mid-term

This section considers the mid-term collection. Figure 14 reports the box plots of AP and nDCG for each system on the mid-term collection.

The considerations for the mid-term collection are largely in line with those made for the short-term. As in the previous case, all systems exhibit similar distribution shapes, with comparable interquartile ranges and whisker lengths, and full coverage of the score range from 0.0 to 1.0. In both AP and nDCG plots, the mean values consistently exceed the medians, indicating a right-skewed distribution due to a subset of high-performing queries.

A notable difference, however, is that System 3 and System 5 exhibit slightly higher central values, particularly in the nDCG plot. This hints at a marginally stronger performance for these two systems

Table 14
Tukey HSD test AP (short-term collection)

System A	System B	P-value
System 1	System 2	**7.51e-16**
System 1	System 3	**0**
System 1	System 4	**1.26e-06**
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	**0.0214**
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**0.0251**
System 4	System 5	**0**

Table 15
Tukey HSD test nDCG (short-term collection)

System A	System B	P-value
System 1	System 2	**1.01e-06**
System 1	System 3	**0**
System 1	System 4	**0.0019**
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	0.4938
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**0.0010**
System 4	System 5	**0**

on the mid-term collection, even though the overall variability remains similar across all systems.

As done for the short-term collection, we report the tables with the results of the ANOVA2 tests conducted on AP and nDCG. In this case as well, both p-values associated with the systems and the queries fall below the significance threshold α. Consequently, we reject the null hypothesis and proceed with the *Tukey's HSD test*, reporting the plots obtained.

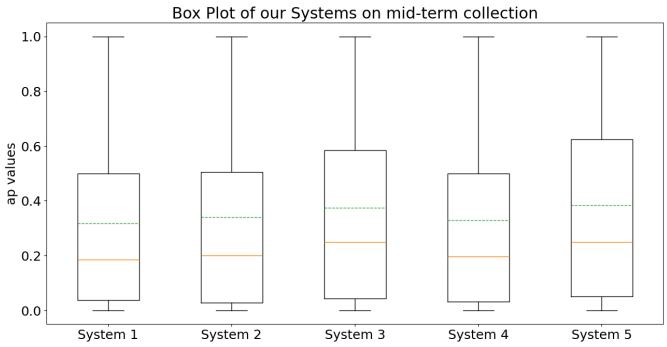
Table 16
ANOVA2 AP on mid-term collection.

Source	SS	df	MS	F	Prob>F
Columns	23.8461	4	5.9615	289.3370	2.24e-244
Rows	4027.2542	7199	0.5594	27.1509	0
Error	593.3145	28796	0.0206		
Total	4644.4147	35999			

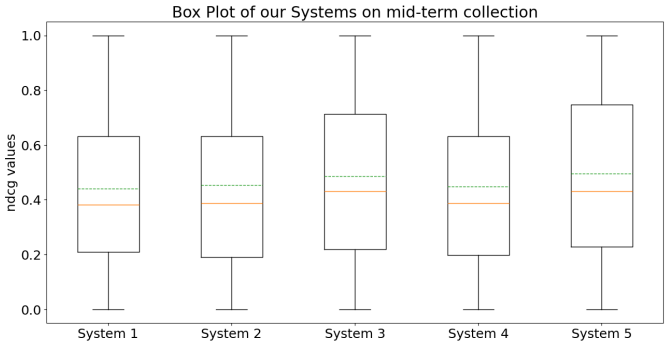
Table 17
ANOVA2 nDCG on mid-term collection.

Source	SS	df	MS	F	Prob>F
Columns	16.7296	4	4.1824	314.0481	6.08e-265
Rows	3274.2270	7199	0.4548	34.1514	0
Error	383.4960	28796	0.0133		
Total	3674.4526	35999			

In the mid-term scenario (Figure 15), we observe a pattern of differences among the systems that closely mirrors what emerged in the short-term evaluation. Here, System 1 has been chosen as the



(a) AP values



(b) nDCG values

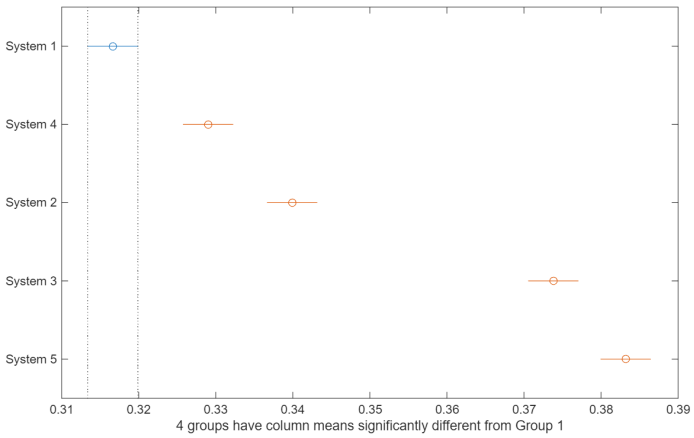
Figure 14: Box plots of the five systems evaluated on the mid-term collection using AP and nDCG.

reference group as well. As before, each system's mean performance is plotted with its confidence interval, allowing us to judge at a glance which differences are genuine and which might arise by chance. Just as in the short-term case, four out of the five systems differ significantly from the reference at $\alpha = 0.05$: none of the confidence intervals for Systems 2, 3, 4 or 5 overlap with that of System 1, so we can reject the null hypothesis of equal means for all those pairwise comparisons.

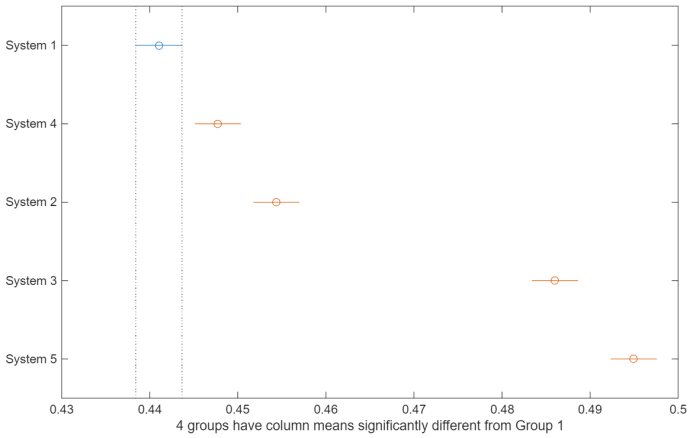
As before, we report the output of the test in Table 18 and Table 19 for AP and nDCG, respectively.

Table 18
Tukey HSD test AP (mid-term collection)

System A	System B	P-value
System 1	System 2	**7.77e-23**
System 1	System 3	**0**
System 1	System 4	**2.37e-06**
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	**4.97e-05**
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**0.00083**
System 4	System 5	**0**



(a) AP values



(b) nDCG values

Figure 15: Plot of the differences in means across multiple groups, illustrating the variation within and between groups in the mid-term collection.

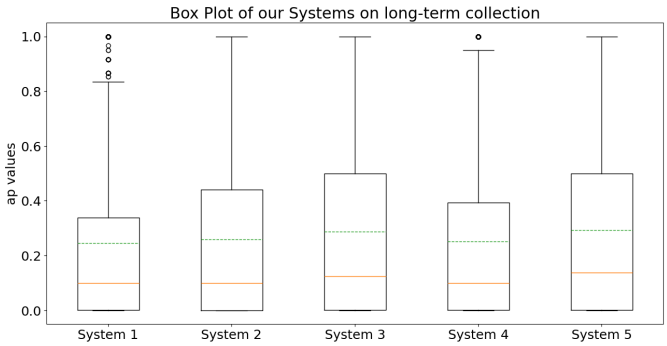
Table 19
Tukey HSD test nDCG (mid-term collection)

System A	System B	P-value
System 1	System 2	**3.53e-11**
System 1	System 3	**0**
System 1	System 4	**0.0046**
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	**0.0049**
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**3.29e-05**
System 4	System 5	**0**

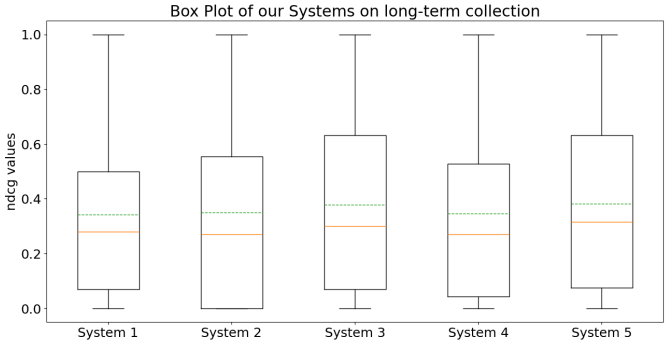
6.3. Long-term

Lastly, we discuss the long-term collection. In Figure 16 a marked difference is observed in the medians, with System 1 and System 4 clearly underperforming relative to the others. Their boxes are more compressed towards the bottom of the scale, indicating consistently low performance across topics.

Moreover, in Figure 16a System 1 exhibits a greater number of outliers, suggesting instability or sporadic good performance on a few topics, but generally poor results overall. System 2 also shows worse performance compared to the other systems, though not as poor as System 1 and System 4.



(a) AP values



(b) nDCG values

Figure 16: Box plots of the five systems evaluated on the long-term collection using AP and nDCG.

With respect to ANOVA2 results in Table 20 and Table 21, we can draw analogous conclusions as for the short-term and mid-term collections: both factors exhibit extremely low p-values, indicating that differences among retrieval systems and the inherent variability across queries are highly significant. Consequently, we apply *Tukey's Honestly Significant Difference* post-hoc test.

Table 20
ANOVA2 AP on long-term collection.

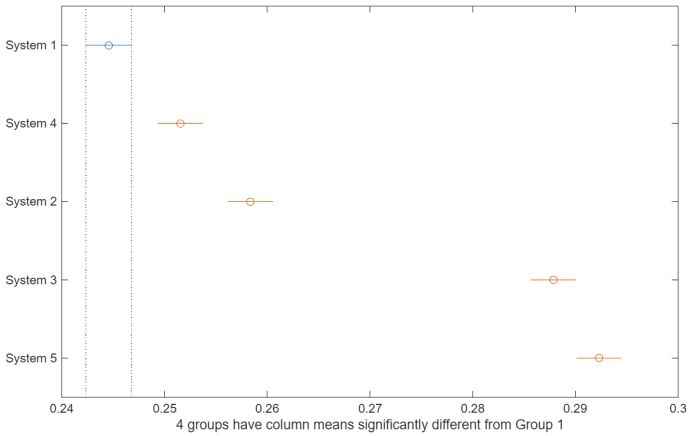
Source	SS	df	MS	F	Prob>F
Columns	21.7984	4	5.4496	361.0118	1.23e-306
Rows	5720.4241	11519	0.4966	32.8980	0
Error	695.5343	46076	0.0151		
Total	6437.7568	57599			

For the long-term collection, the results of *Tukey's HSD* test (Figure 17) reveal patterns consistent with those observed in the short-term and mid-term evaluations. System 1 is again used as the reference group. In the AP comparison (Figure 17a), the confidence intervals for Systems 2, 3, 4, and 5 do not overlap with that of System 1, indicating statistically significant differences in mean performance. This confirms that, as in the previous evaluations, System 1 performs significantly differently from all other systems at the $\alpha = 0.05$ significance level. In contrast, for the nDCG metric (Figure 17b), only Systems 2, 3, and 5 exhibit confidence intervals that do not intersect with that of System 1, suggesting significant

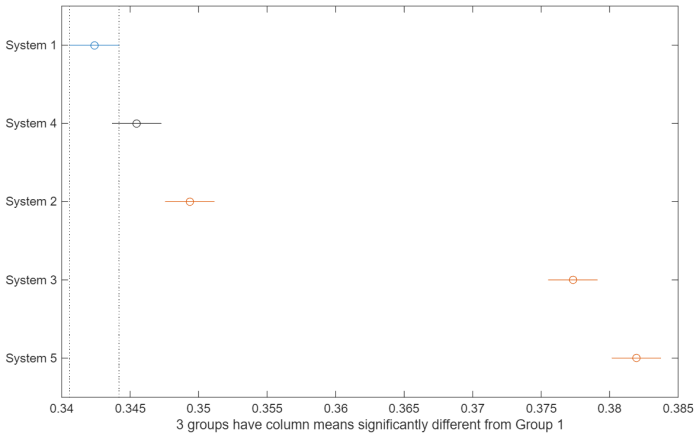
Table 21
ANOVA2 nDCG on long-term collection.

Source	SS	df	MS	F	Prob>F
Columns	16.2797	4	4.0699	403.9727	0
Rows	5730.4111	11519	0.4975	49.3783	0
Error	464.2052	46076	0.0101		
Total	6210.8960	57599			

differences in these cases. System 4, on the other hand, does not show a statistically significant difference from System 1 in terms of nDCG, as also confirmed by the corresponding p-value reported in Table 23, which exceeds the significance threshold of $\alpha = 0.05$.



(a) AP values



(b) nDCG values

Figure 17: Plot of the differences in means across multiple groups, illustrating the variation within and between groups in the long-term collection.

Table 22
Tukey HSD test AP (long-term collection)

System A	System B	P-value
System 1	System 2	**5.42e-17**
System 1	System 3	**0**
System 1	System 4	**0.00016**
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	**0.00024**
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**0.0456**
System 4	System 5	**0**

Table 23
Tukey HSD test nDCG (long-term collection)

System A	System B	P-value
System 1	System 2	**1.38e-06**
System 1	System 3	**0**
System 1	System 4	0.1331
System 1	System 5	**0**
System 2	System 3	**0**
System 2	System 4	**0.0281**
System 2	System 5	**0**
System 3	System 4	**0**
System 3	System 5	**0.0042**
System 4	System 5	**0**

7. Conclusions and Future Work

This section summarizes the different techniques that we explored in order to improve our Information Retrieval system. Starting from a basic configuration made by using Lucene's API, we tested its performance to set a reference score, and we discovered how different data can greatly change the performance of our system. We explored several possible configurations for our analyzer, including a lemmatization step of the entire collection, and we discovered that using a light stemming process produces a higher score than using more complex stemmers. We then moved on with testing different approaches to improve the matching and the ranking of the documents: we first introduced a different way to index the documents, by splitting them into smaller pieces and scoring each piece individually, but we soon discovered that this approach leads our system to underperform with respect to the fixed baseline.

We later moved on into expanding the query terms using synonyms, which did not lead to an improvement of the system, to the releasing of terms' positions in the proximity search approach, which instead proved to increase the effectiveness of the system, especially when fine-tuning the slop parameter. We also explored the possibility of using the top-most retrieved documents to improve the query itself through the pseudo-relevance feedback technique, especially useful when precision is high at the top-most ranking positions. Finally, we used sentence embeddings to rerank the results returned by our system through the Roberta-Large model, which again leads us to a better improvement of our system's performance.

When all these techniques are deployed individually, it may appear difficult to understand if the system is improving, however combining them all appears not to be the most effective way to increase

the performance of the system, which is also very sensitive to data that are fed as input. In order to continue improving the efficiency of an IR system, more research is needed. With regard to synonyms and query expansion, the most intuitive way to improve the query is to use a language model to learn the synonyms directly by the collection, instead of relying on external datasets. As regards the lemmatization process, it may be useful to construct a dictionary of arguments and use the same dictionary to tag each provided query with a more general word and use those to improve the pool of matched documents. Another possible improvement of the system could be made by joining together similar queries, so that the user may find more documents related to the possibly imprecise query he submitted. Once again, the huge development of LLMs that we are seeing nowadays can be a great tool to improve the effectiveness of those IR systems, especially when the data collections are mostly made up of textual documents.

8. Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Cancellieri, A. El-Ebshihy, T. Fink, P. Galuščáková, G. Gonzalez-Saez, L. Goeuriot, D. Iommi, J. Keller, P. Knoth, P. Mulhem, F. Piroi, D. Pride, P. Schaer, Overview of the CLEF 2025 LongEval Lab on Longitudinal Evaluation of Model Performance, in: J. Carrillo-de Albornoz, J. Gonzalo, L. Plaza, A. García Seco de Herrera, J. Mothe, F. Piroi, P. Rosso, D. Spina, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Sixteenth International Conference of the CLEF Association (CLEF 2025), 2025.

[2] Sushilkumar Chavhan, M. Raghuwanshi, R. Dharmik, Information Retrieval using Machine learning for Ranking: A Review , https://iopscience.iop.org/article/10.1088/1742-6596/1913/1/012150/meta, 2021.

[3] R. Nogueira, W. Yang, K. Cho, J. Lin, Multi-stage document ranking with bert, 2019. URL: https://arxiv.org/abs/1910.14424. arXiv:1910.14424.

[4] A. Basaglia, A. Stocco, M. Popovic, N. Ferro, Seupd@clef:team dam on reranking using sentence embedders, 2024. URL: https://ceur-ws.org/Vol-3740/paper-216.pdf .

[5] R. Pramana, Debora, J. J. Subroto, A. A. S. Gunawan, Anderies, Systematic literature review of stemming and lemmatization performance for sentence similarity, in: 2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA), IEEE, 2022, pp. 1–6. doi:10.1109/icitda55840.2022.9971451.

[6] V. Balakrishnan, E. Lloyd-Yemoh, Stemming and lemmatization: A comparison of retrieval performances, 2014. URL: https://api.semanticscholar.org/CorpusID:52998253.

[7] K. Abedini, A. Akysh, A. Fahoud, Seupd@clef: Team kalu on improving search engine performance with query expansion and re-ranking approach, 2024. URL: https://ceur-ws.org/Vol-3740/paper-214.pdf .

[8] I. A. H. Chen, J. Moncada-Ramírez, N. Santini, G. Zago, N. Ferro, Seupd@clef: Team jihuming on enhancing search engine performance with character n-grams, query expansion, and named entity recognition, 2023. URL: https://ceur-ws.org/Vol-3497/paper-185.pdf .

[9] The Apache Software Foundation, Apache Lucene, https://lucene.apache.org/, 2025.

[10] The Apache Software Foundation, Apache OpenNLP, https://opennlp.apache.org/, 2025.

[11] B. Sagot, The lefff, a freely available and large-coverage morphological and syntactic lexicon for french, 7th international conference on Language Resources and Evaluation (LREC 2010) (2010).

[12] Coulomb, Claude, A French Lemmatizer in Python based on the LEFFF, https://github.com/ClaudeCoulombe/FrenchLefffLemmatizer, 2017.

[13] W. Yin, H. Schütze, MultiGranCNN: An architecture for general matching of text chunks on multiple levels of granularity, in: C. Zong, M. Strube (Eds.), Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China, 2015, pp. 63–73. URL: https://aclanthology.org/P15-1007/. doi:10.3115/v1/P15-1007.

[14] B. Sagot, D. Fišer, Building a free french wordnet from multilingual resources, 2008.

[15] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL: https://aclanthology.org/D19-1410.

[16] N. Reimers, I. Gurevych, sentence-transformers/all-roberta-large-v1, https://huggingface.co/sentence-transformers/all-roberta-large-v1, 2020.