# SEUPD@CLEF: Team Searchill

Alessandro Di Frenna[1], Andrea Di Tillo[1], Luca Pellegrini[1], Marco Tomaiuoli[1] and Artur Voit-Antal[1]

[1]*University of Padua, Italy*

**Abstract**

This report describes the search engine developed by the team "Searchill" for the LongEval-Retrieval Task 1 at the CLEF 2025 conference. The main objective of the project was to develop an effective yet efficient search engine. To achieve this goal, our team has extensively experimented with various techniques and approaches. To assess system performance, we used the CLEF corpus and human judgements. Our top-performing system, which combines traditional methods with AI, has encouraging results and might be the basis for further advancements in the field.

**Keywords**

CLEF 2023, LongEval, Information Retrieval, Search Engine, Natural Language Processing

## 1. Introduction

The amount of information available online has been rapidly increasing due to the internet's fast growth, making it difficult to find reliable and pertinent information. Search engines are essential in this situation as they allow users to quickly and easily locate accurate information. Information retrieval systems are now essential in almost all the fields. Our study investigates the complexities of information retrieval and how it can be enhanced, as well as how well they can adjust to the changing digital environment.

The goal of this research is to create a reliable retrieval system that can handle the ever-changing nature of the internet. We do this by using the Longeval Websearch collection [1], an extensive dataset of online pages, topics, and user interactions. As a result of this effort, our understanding of information retrieval systems in the context of search engines has greatly improved. In particular, we focused on improving query and document processing methods to maximise ranking outcomes and provide users with the most relevant information.

The paper is organized as follows: Section 2 describes our approach; Section 3 explains our experimental setup; Section 4 discusses our main findings; finally, Section 5 draws some conclusions and outlooks for future work.

### 1.1. Related Work

To begin addressing the project requirements, we conducted a thorough analysis of the specifications and file formats necessary for implementation. We relied on materials provided by the CLEF LongEval [1] website and the accompanying 2023 documentation [2] to identify the essential elements needed to organize our workflow effectively. Guided by the tutoring sessions from the SearchEngines course at the University of Padua, we proceeded with the development of the key components—indexer, analyzer, and searcher—following the framework introduced and discussed in class. For implementing the re-ranker, we employed the API offered and documented by Cohere [3].

*"Search Engines", course at the master degree in "Computer Engineering", Department of Information Engineering, and at the master degree in "Data Science", Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy. Academic Year 2024/2025*

Alessandro.DiFrenna@studenti.unipd.it (A. D. Frenna); Andrea.DiTillo@studenti.unipd.it (A. D. Tillo); Luca.Pellegrini@studenti.unipd.it (L. Pellegrini); Marco.Tomaiuoli@studenti.unipd.it (M. Tomaiuoli); Artur.VoitAntal@studenti.unipd.it (A. Voit-Antal)

## 2. Methodology

In this section, we describe the methodology adopted to develop our IR system for the task.

### 2.1. Parser

The parser is responsible for reading and extracting structured data from complex document formats, enabling effective indexing and retrieval. Our document parsing framework consists of several components designed to handle different document structures efficiently:

- **DocumentParser:** This abstract class serves as the core of our parsing framework. It provides a template for iterating over documents. The class ensures that documents are read correctly.
- **ParsedDocument:** This class encapsulates the structure of a parsed document. It defines essential fields such as:
  - `ID` - A unique identifier for each document.
  - `Body` - The main content of the document.
  - `Start` - Initial lines of the document, often used for quick reference.
  - `Highlights` - Notable words or phrases extracted from tags or special formatting, enhancing keyword extraction.
- **TrecParser:** This specialized class extends `DocumentParser` to handle TREC-formatted documents. Key functions include:
  - Regex-based extraction of document fields such as `<DOCNO>` and `<DOCID>`.
  - Management of document content using a multistage buffering approach to efficiently handle large text bodies.
  - Extraction of emphasized content through patterns such as `hashtags` and `strong` tags, improving the semantic richness of stored highlights.
  - Support for emoji and URL filtering, ensuring text data is clean and normalized before analysis.
- **Error Handling:** Robust error management is implemented to gracefully handle missing or malformed data, maintaining the stability and reliability of the parser.

This structured parsing methodology allows for the seamless transformation of raw text into well-defined data objects, paving the way for accurate indexing and retrieval operations.

### 2.2. Analyzer

The `FrenchAnalyzer` is designed to process French texts, integrating multiple components to ensure accurate analysis:

- **Tokenization:** We provide multiple tokenizer options to cater to different text handling needs:
  - `WhitespaceTokenizer` - Splits texts at whitespace, ideal for structured input.
  - `LetterTokenizer` - Separates tokens at non-letter characters, useful for handling alphabetic text.
  - `StandardTokenizer` - Uses advanced parsing rules for handling more complex textual structures, such as handling punctuation and special characters. Includes Named Entity Recognition (NER) for recognising email addresses, URLs, and numeric sequences as single tokens.
  - `OpenNLPTokenizer` - Uses OpenNLP statistical models for tokenization, which is beneficial for handling complex language.

- **Character Folding:** Using `ICUFoldingFilter`, characters are normalized to their basic forms, standardizing variations such as accented characters to their base, which improves the uniformity of the processing.
- **Elision Removal:** The `ElisionFilter` targets contracted French words, removing apostrophes and similar contractions, treating contractions like "l'histoire" simply as "histoire".
- **Abbreviation Expansion:** The `AbbreviationExpansionFilter` employs a predefined map to expand abbreviations, ensuring that terms such as "Dr." are expanded to "Doctor", further improving processing uniformity.
- **POS Tagging and Compound Formation:** Through `CompoundPOSTokenFilter`, part-of-speech tags are assigned using the OpenNLP POS model. This allows for recognising compound words as such and treating them as single tokens.
- **N-grams Generation:** `ShingleFilter` is optionally applied to produce n-grams that capture context within a specified window size.
- **Stopword Removal:** The `StopFilter`, loaded with a list of irrelevant tokens, removes common stopwords that do not contribute to the semantic meaning of texts and queries.
- **Length Filtering:** The `LengthFilter` ensures that only tokens within a specified character range are retained, filtering out noise and exceedingly frequent short tokens.
- **Position Increment Adjustment:** Implemented via `PositionFilter`, this adjusts the `positionIncrement` attribute of tokens, maintaining consistent positional data for proximity-based queries.
- **Stemming and Lemmatization:** customised to handle French morphology:
  - `FrenchLightStemFilter` - Applies a light stem, suitable for texts requiring minimal conversion.
  - `FrenchMinimalStemFilter` - Executes minimal stemming, avoiding over-reduction.
  - `SnowballFilter`(snow) - Employs the Snowball stemming algorithm, recognized for its effectiveness across languages.
  - `OpenNLPLemmatizerFilter` - A lemmatization approach that uses OpenNLP to convert words to their base forms, capturing the underlying semantics.

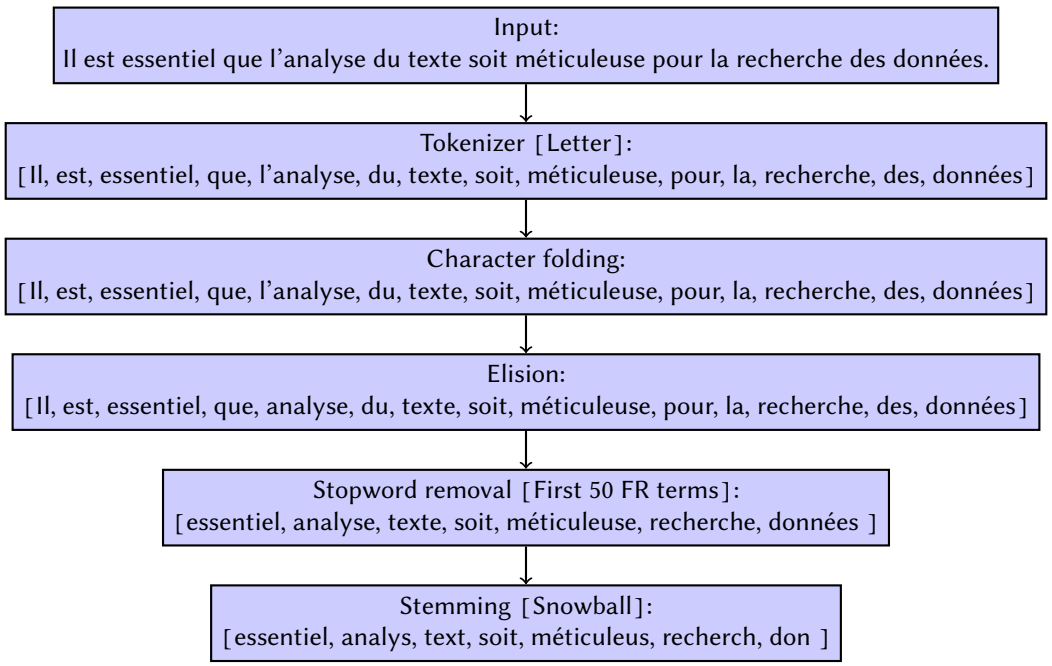


**Figure 1:** Example of the analyzer process.

## 2.3. Indexer

- **BodyField**:
  - **BodyField**: represents the body field of a document:
    - **token**: the body is broken into tokens.
    - **frequency**: we save the frequency of the tokens.
    - **position**: we save the position of the tokens, this allows us to use phrase queries.
    - **body store**: we save the body of the documents, this allows us to pass the document's bodies to a re-ranker.
  - **StartField**:
    - **token**: the start is broken into tokens.
    - **frequency**: we save the frequency of the tokens.
    - **body store**: we save the start of the documents.
  - **HighlightedField**:
    - **token**: the hightlighted is broken into tokens.
    - **frequency**: we save the frequency of the tokens.
    - **body store**: we save the highlights of the documents.
- **DirectoryIndexer**:
  This class indexes documents found within a designated directory. It allows for several parameters, such as the directory where the documents are stored, the type of parser used to parse the documents, the analyzer used to process the text, the similarity metric employed for indexing, the anticipated number of documents, and the location where the resulting index will be saved.

| Collection | Docs Size (GB) | Body Terms | Number of Documents | Index Size (GB) |
|---|---|---|---|---|
| French 2022-06 | 9,3 | 3,423,470 | 1,590,024 | 7.5 |
| French 2023-01 | 12.8 | 38,313,777 | 2,537,554 | 12.4 |

**Table 1**
Indexing output details.

### 2.3.1. StopList Choice

After we have indexed all the documents part of the training set, we used LUKE, a tool provided by Lucene that allowed us to inspect our indexed documents and see how the documents were indexed and identify the most frequent tokens in our collection.

After indexing all the French documents without using stop lists or Stemmers, we observed the first lines of results.

We noticed there are English terms and numbers with significant frequency values alongside French terms. Based on this observation, we decided to create various stop lists incorporating different combinations of numbers, English terms, or both. We then examined the computational results of these configurations. All this was done to verify the importance that these elements have in the context of the documents analyzed.
The created stop lists are:

- **stopList40FR-10EN :** It contains the first 40 French terms and 10 English terms.
- **stopList40FR-10NUM :** It contains the first 40 French terms and 10 numbers.
- **stopList35FR-10EN-5NUM :** It contains the first 35 French terms, 10 english words and 5 numbers.

- **stopList50FR-2022-06 :** It contains the first 50 French terms from the French collection 2022-06.
- **stopList50FR-2023-01 :** It contains the first 50 French terms from the French collection 2023-01.

We chose the best one, and decided to use it for all the runs.

- **Stoplists tuning:**

| Run name 2022-06 | nDCG | MAP |
|---|---|---|
| stoplist-BstoplistFR | 0,1682 | 0.0896 |
| stoplist-50FR | 0,1681 | 0.0897 |
| stoplist-40FR-10N | 0,1673 | 0.0890 |
| stoplist-40FR-10EN | 0,1694 | 0.0902 |
| stoplist-35FR-10EN-5N | 0,1679 | 0.0894 |

**Table 2**
Stoplist tuning table 2022-06.

| Run name 2023-01 | nDCG | MAP |
|---|---|---|
| stoplist-BstoplistFR | 0,2540 | 0.1620 |
| stoplist-50FR | 0,2545 | 0.1624 |
| stoplist-40FR-10N | 0,2530 | 0.1619 |
| stoplist-40FR-10EN | 0,2587 | 0.1631 |
| stoplist-35FR-10EN-5N | 0,2531 | 0.1622 |

**Table 3**
Stoplist tuning table 2023-01.

- **Stemmer tuning:**

| Run name 2022-06 | nDCG | MAP |
|---|---|---|
| stem-FrenchLight | 0.1398 | 0.0703 |
| stem-SnowBall | 0.1499 | 0.0767 |
| stem-FrenchMinimal | 0.1426 | 0.0722 |

**Table 4**
Stemmer tuning 2022-06

| Run name 2023-01 | nDCG | MAP |
|---|---|---|
| stem-FrenchLight | 0.2207 | 0.1345 |
| stem-SnowBall | 0.2315 | 0.1422 |
| stem-FrenchMinimal | 0.2250 | 0.1370 |

**Table 5**
Stemmer tuning 2023-01

- **Tokenizer tuning:**

| Run name 2022-06 | nDCG | MAP |
|---|---|---|
| tokenizer-WhiteSpace | 0.1541 | 0.0796 |
| tokenizer-Letter | 0.1723 | 0.0923 |
| tokenizer-Standard | 0.1710 | 0.0919 |

**Table 6**
Stemmer tuning 2022-06

| Run name 2023-01 | nDCG | MAP |
|---|---|---|
| tokenizer-WhiteSpace | 0.2420 | 0.1512 |
| tokenizer-Letter | 0.2675 | 0.1710 |
| tokenizer-Standard | 0.2656 | 0.1695 |

**Table 7**
Stemmer tuning 2023-01

## 2.4. Searcher

Within the searcher component, a Boolean query strategy was employed to facilitate query expansion and enhance the overall effectiveness of the information retrieval system. The initial query terms were designated as mandatory for document retrieval, while the expanded terms were incorporated as optional elements, thereby allowing for greater flexibility in matching relevant documents. Furthermore, a boosting mechanism was applied to assign increased weight to the original query, reinforcing its central role in the retrieval process. The subsequent section provides a detailed account of the query expansion methodology implemented.

### 2.4.1. Start

It is commonly observed that the core subject matter of a document is often introduced at the beginning of the text. Building upon this assumption, we implemented a strategy within the indexer to store the first five lines of each document, leveraging the structural layout provided by the LongEval dataset in TREC format. This design choice allowed the search engine to perform query operations specifically within this initial segment — termed the `start` — rather than scanning the entire document body.

To further enhance retrieval precision, we assigned greater weight to query matches occurring within the `start` section compared to those found elsewhere in the document. The underlying rationale was that documents introducing the query terms early on are more likely to align with the user's search intent. This emphasis on the initial portion was intended to improve both the relevance and the overall quality of the retrieved results.

### 2.4.2. N-grams

As an initial step in our query expansion strategy, we employed word N-grams—sequences of adjacent terms extracted from the original query—to enrich the query representation. This process involves tokenizing the original query into individual words and subsequently recombining them into contiguous word pairs. For example, the query "`running shoes women`" is tokenized into "`running`", "`shoes`", and "`women`", from which we generate the bi-grams "`running shoes`" and "`shoes women`".

While this technique has the potential to enhance retrieval accuracy by capturing meaningful term associations, we deliberately assigned a lower weight to these expanded terms within the Boolean query formulation. This decision was made to prioritize documents that more closely align with the original query, thus preserving its semantic integrity.

Moreover, we restricted the generation of N-grams to a maximum of two words. In our view, extending the N-gram length beyond this threshold would introduce an excessive number of query variants, potentially diluting the relevance of the results and increasing computational overhead.

### 2.4.3. Fuzzy Search

To further improve the robustness of our search system, we integrated a fuzzy search mechanism capable of handling approximate matches between query terms and document content. This technique is particularly effective in addressing minor input errors such as typographical mistakes, letter transpositions, or common abbreviations. By setting a similarity threshold, the system ensures that only plausible variations of the query terms are considered, thereby maintaining an acceptable level of accuracy. For example, a query such as "`Adibas chaussures`" may successfully retrieve documents containing the correctly spelled brand name "`Adidas`".

The fuzzy searcher leverages the Damerau–Levenshtein distance as its core similarity metric. This algorithm—also referred to as the optimal string alignment distance—quantifies the dissimilarity between two strings based on the minimum number of edit operations (insertions, deletions, substitutions, or adjacent transpositions) required to transform one string into the other.

In our implementation of fuzzy search [4], candidate terms are collected and scored according to their respective edit distances, with only the highest-ranking terms incorporated into the expanded query. Importantly, we deliberately avoided using high distance thresholds, as such configurations tend to generate overly broad matches, ultimately reducing retrieval precision and degrading overall system performance.

**Fuzzy Tuning**

To assess the effectiveness of incorporating fuzzy queries within our Boolean query framework, we employed an empirical evaluation strategy. Specifically, we compared the performance of a baseline configuration — where no query expansion techniques were applied and only the original query was used — against a series of runs integrating fuzzy queries with varying weight parameters. This process, which we refer to as `Fuzzy Tuning`, aimed to isolate and measure the contribution of fuzzy matching

to retrieval performance. Notably, all experiments were conducted without the use of the re-ranking component.

We determined that a weight increment of 0.1 constituted a suitable compromise between granularity and experimental feasibility, as finer increments (e.g., 0.05) would have necessitated a disproportionately high number of runs without a commensurate gain in insight.

As illustrated in Table 5, the optimal performance was observed when the fuzzy component was assigned a weight of 0.1. This result is consistent with the intended role of fuzzy filtering: while it is effective at capturing minor orthographic variations (e.g., typographical errors or alternate spellings), overemphasizing its contribution may inadvertently amplify terms that are lexically similar yet semantically unrelated. Consequently, excessive weighting can degrade the quality of the retrieved documents by introducing noise into the results.

#### 2.4.4. Dictionary

During the analysis of user input, we observed that certain queries appeared in unconventional formats, such as URLs, dot-separated terms (e.g., `term1.term2.term3`), or concatenated words without spacing (e.g., `term1term2term3`), which posed challenges for effective processing by the search engine. To mitigate issues caused by punctuation, we implemented a function based on regular expressions to systematically remove such characters from the input.

In addressing the problem of concatenated terms, we employed a dictionary derived from the provided synonyms file. This dictionary enables the system to identify and extract meaningful subterms embedded within single-word queries. Notably, this approach was applied exclusively to queries composed of a single token, as extending it to multi-word queries introduced considerable computational overhead and negatively impacted system performance.

The motivation for focusing on single-word queries lies in the higher likelihood of such inputs being affected by typographical errors. In these cases, suggesting alternative or similar terms can significantly enhance retrieval quality. Conversely, in multi-word queries, the probability of all constituent terms being misspelled is low; thus, attempting to alter each term could reduce both precision and overall system effectiveness.

#### 2.4.5. Synonyms

Implementing synonyms in retrieval poses challenges, including semantic ambiguity and noise due to polysemy and irrelevant matches. While synonyms theoretically broaden search scope, actual gains depend on context-specific relevance. To mitigate computational overhead, synonyms were managed at query time using a query expansion model. This approach aims to enhance retrieval accuracy while maintaining system efficiency.

Consider the example query "`Chat orange`", normally, the information retrieval (IR) system first parses and analyzes the query and searches for "`orange`" and "`chat`". However, if a relevant document contains "`gingembre`" instead of "`orange`" and "`félin`" instead of "`chat`", the document will not be considered relevant. The searcher takes the original query, searches a synonym collection for synonyms of those words, and retrieves three synonyms. If there are more than three available in the collection, three synonyms are chosen randomly, and these are then added to the Boolean query. We chose to use a synonym collection sourced from GitHub [5].

**Synonyms Tuning**: Since the synonyms approach involves selecting three random synonyms from the collection, we chose not to adopt an experimental approach as we did with the Fuzzy and Phrase tools, due to a lack of repeatability stemming from the aforementioned randomness. We assigned a weight of 0.1 to each synonym to maintain the choices made for the Phrase and Fuzzy Query tools.

### 2.4.6. Phrase Query

To further enhance the precision of our retrieval system, we implemented a **Phrase Query Searcher** designed to identify documents containing a specific sequence of terms in the exact or near-exact order in which they appear in the query. Phrase queries require that the specified terms occur in contiguous positions within the document. However, to allow for a degree of flexibility in term positioning, we employed the *slop* parameter, which defines the maximum number of positional shifts permitted between the terms.

This mechanism is particularly beneficial in accommodating natural variations in language, such as minor rearrangements or the insertion of additional words. For example, a query for the phrase "quick brown fox" with a slop value of 1 would also match documents containing "quick fox brown," thereby increasing recall without significantly deviating from the original query intent.

The incorporation of phrase queries thus improves the search engine's ability to retrieve documents that closely align with the semantic structure of the user input, even when exact word order is not preserved.

In the context of LongEval 2025 [1] the official queries are provided in two formats. The first follows the TREC format, structured as:

```
<top>
<num>q062228</num>
<title>aeroport bordeaux</title>
</top>
```

Here, the `<num>` tag denotes the query identifier, while the `<title>` tag contains the textual content of the query [**?** ]. The alternative format, a TSV (tab-separated values) representation, is structured as:

```
62228 aeroport Bordeaux
```

In this case, the query identifier and the query text are separated by whitespace [6].

### 2.4.7. Word2Vec

To provide word similarity functionality through a web-accessible interface, we developed a lightweight HTTP server using the Flask framework, chosen for its simplicity and ease of deployment in small-scale applications. At the core of the system lies the `KeyedVectors` interface from the Gensim library, which allows efficient loading and querying of pre-trained word embedding models.

For our implementation, we employed the 300-dimensional French fastText word vectors (cc.fr.300.vec), which were trained on a large-scale Common Crawl corpus. These vectors are particularly suitable for capturing semantic relationships between words in French, thanks to their subword-based architecture that improves coverage and performance on rare or morphologically rich terms.

The application exposes a single RESTful endpoint, `/similar`, designed to handle HTTP GET requests. Clients can query this endpoint by passing a word as a query parameter. To ensure consistent and reliable behavior, especially in the face of spelling variants and orthographic inconsistencies, the input word is first normalized: it is converted to lowercase and stripped of any diacritics using the unidecode library. This preprocessing step helps enhance the robustness of the similarity computation, enabling more tolerant matching and improved usability in real-world scenarios.

The main purpose of this service is to support query expansion within our information retrieval pipeline. When a user submits a search query, the system sends the main term to the `/similar` endpoint. The endpoint responds with a list of semantically related words retrieved from the fastText embeddings. These similar terms are then automatically appended to the original query, effectively expanding it. This strategy enhances recall by allowing the retrieval system to consider contextually related terms, improving the chances of retrieving relevant documents even when exact matches are not present.

### 2.4.8. LLM

LLM-based query expansion is a technique used to enhance the effectiveness of a search engine by automatically expanding the original query formulated by the user. This methodology is particularly useful in cases where the user input is too generic, vague, or ambiguous, and therefore insufficient to return truly relevant results.

In our case, the query is sent to a Large Language Model (LLM), specifically *ChatGPT-3.5-turbo*, which is responsible for interpreting the meaning of the sentence and generating three to four related terms.

This number of terms was chosen strategically: an expansion that is too broad would introduce informational noise, potentially reducing the precision of the results, while an expansion that is too narrow would compromise the overall effectiveness of the technique.

We therefore opted for a balanced solution, capable of enriching the query in a meaningful yet controlled way. These terms are not simple synonyms, but semantically related concepts that help enrich the context of the search. The goal is to provide the search engine with additional cues to identify relevant documents, even when they do not contain the exact terms used in the original query.

The choice of using ChatGPT-3.5-turbo was driven by a trade-off between performance and cost: although less powerful and accurate than more advanced models such as GPT-4o or GPT-4.1, it proved to be sufficiently effective for our needs, at a significantly lower price compared to its counterpart.

### 2.4.9. Query Construction

The core search process employs a dynamic Boolean query builder with weighted clauses:

- **Base Query**: Mandatory clause parsed from the original query string
- **Field Boosting**:
  - Start field matches boosted by 1.5×
  - Highlight field matches boosted by 1.2×
- **Query Expansion**: Optional features:
  - Dictionary-based synonyms (0.1× weight)
  - Fuzzy search with edit distance=2 (0.2×-0.8× weights), which allows for typos
  - Phrase queries with slop=5 window

### 2.4.10. Query Expansion Techniques

We implemented six complementary expansion strategies:

#### N-grams Generation

- Uses ShingleFilter with default 2-gram window
- Applies 0.3× weight to generated bigrams
- Filters single-word n-grams post-generation

#### Synonym Expansion

- Loads synonym mappings
- Distributes weights equally among synonyms
- Randomly selects 3 synonyms per term when multiple exist

#### Neural Expansion

- LLM-based term generation via GPT-3.5/DeepSeek APIs
- Requests 5-10 related terms per query
- Applies 0.4× weight to generated terms

**Pseudo-Relevance Feedback (PRF)**

- Extracts top 20 terms from initial results
- Re-searches with expanded 0.5× weighted terms
- Filters short tokens (<3 characters)

#### 2.4.11. Re-ranking Pipeline

Final results undergo neural re-ranking:

1. Normalize initial scores to [0,1] range
2. Retrieve top 100 document bodies
3. Invoke Cohere multilingual-v3 re-ranker
4. Combine scores using linear interpolation:

$$finalScore = 0.4 \cdot BM25 + 0.6 \cdot CohereScore$$

#### 2.4.12. BM25

For document scoring, we adopted the **BM25** ranking algorithm [7], a well-established probabilistic retrieval model known for its effectiveness in information retrieval tasks. BM25 evaluates the relevance of documents based on term frequency, inverse document frequency, and document length normalization.

A notable limitation of BM25 lies in its sensitivity to parameter tuning, as identifying the optimal values for its parameters (typically $k_1$ and $b$) can be a non-trivial task. To address this, we opted to employ the default parameter settings provided by the Lucene library. This choice was guided by the consideration that manual parameter optimization might lead to overfitting on the development dataset, thereby impairing the model's generalization capability on previously unseen data.

## 3. Experimental Setup

The experiments were conducted on a MacBook M1 Pro with the following hardware specifications:

- **CPU:** Apple Silicon M1 Pro, 8-core
- **GPU:** Integrated Apple Silicon M1 Pro, 14-core
- **RAM:** 16 GB LPDDR5, 3200 MHz
- **Storage:** 512 GB SSD with sequential read speeds of up to 4,900 MB/s and write speeds of up to 3,951 MB/s

The datasets utilized in this study are those made available by the LongEval CLEF 2025 Lab, accessible at: https://clef-longeval.github.io/data/

The full implementation of the project can be found at: https://bitbucket.org/upd-dei-stud-prj/seupd2425-searchill/src/master/ — Bitbucket

To employ the Cohere tool for query re-ranking, users must register on the official website (https://cohere.com/) and obtain a free trial API key.

## 4. Results and Discussion

In this Section we present the results obtained by our systems. In particular, Section 4.1 summarizes the system's performance with different configurations on the training dataset. Section 4.2 provides a statistical analysis of the runs submitted to CLEF.

## 4.1. Results on Training Data

We combined the various components outlined in Section 2 and carried out a series of experiments to determine the most effective configuration for our search engine. By systematically tuning and optimizing the parameters of each component, we aimed to achieve the optimal performance. Table 8 presents the Mean Average Precision (MAP) and normalized Discounted Cumulative Gain (nDCG) scores achieved by the different methods, benchmarking their performance on the test documents, queries, and relevance judgments from the TU Wien Research Data Repository [1].

**Table 8**
Evaluation Results for Different Methods (2022 and 2023). Bolded numbers indicate the best stemmer and tokenizer. All "Other Methods" were tested with the baseline of "Snow + Letter + Stoplist (40FR+10EN) + Expansion".

| Method | MAP 2022 | NDCG 2022 | MAP 2023 | NDCG 2023 |
|---|---|---|---|---|
| *Stemmers* | | | | |
| FrenchLight | 0.0703 | 0.1398 | 0.1345 | 0.2207 |
| FrenchMinimal | 0.0722 | 0.1426 | 0.1370 | 0.2250 |
| Snowball | **0.0767** | **0.1499** | **0.1422** | **0.2315** |
| *Tokenizers* | | | | |
| Letter | **0.0923** | **0.1723** | **0.1710** | **0.2675** |
| Standard | 0.0919 | 0.1710 | 0.1695 | 0.2656 |
| Whitespace | 0.0796 | 0.1541 | 0.1512 | 0.2420 |
| *Analyzer methods* | | | | |
| Snow + Letter | 0.0891 | 0.1678 | 0.1620 | 0.2574 |
| Snow + Standard | 0.0882 | 0.1661 | 0.1616 | 0.2564 |
| Snow + Letter + Expansion | 0.0890 | 0.1677 | 0.1623 | 0.2580 |
| Snow + Letter + Stoplist (40FR+10EN) + Expansion | **0.0902** | **0.1694** | **0.1631** | **0.2587** |
| Snow + Letter + Stoplist (40FR+10EN) + Expansion | | | **0.2208*** | **0.3285*** |
| *Other Methods* | | | | |
| Synonyms | 0.0880 | 0.1669 | 0.2169* | 0.3247* |
| Fuzzy | 0.0931 | 0.1740 | 0.2251* | 0.3377* |
| Phrase | 0.0966 | 0.1764 | 0.2509* | 0.3571* |
| PRF | 0.0792 | 0.1581 | 0.1768* | 0.2905* |
| Word2vec | | | 0.2138* | 0.3218* |
| N-grams | 0.0901 | 0.1693 | 0.2209* | 0.3285* |
| Start | 0.0902 | 0.1695 | 0.2210* | 0.3288* |
| Dictionary | 0.0890 | 0.1678 | 0.2190* | 0.3265* |
| Highlights | 0.0901 | 0.1693 | 0.2208* | 0.3285* |
| LLM | 0.0871 | 0.1681 | 0.2100* | 0.3210* |
| Phrase + Fuzzy + Start | 0.0979 | 0.1794 | 0.2571* | 0.3657* |
| Phrase + Fuzzy + Start + Highlights | | | 0.2571* | 0.3657* |
| Phrase + Fuzzy + Start + N-grams | | | 0.2571* | 0.3657* |
| Phrase + Fuzzy + Start + PRF | | | 0.2382* | 0.3502* |
| Phrase + Fuzzy + Start + synonyms | | | 0.2378* | 0.3499* |
| Phrase + Fuzzy + Start + dictionary | | | 0.2569* | 0.3651* |
| Phrase + Fuzzy + Start + N-grams + synonyms + dictionary | | | 0.2383* | 0.3500* |
| Phrase + Fuzzy + Start + Highlights + N-grams | | | 0.2571* | 0.3657* |

Bolded numbers indicate the best stemmer and tokenizer. All "other methods" were tested with the baseline of *Snow + Letter + Stoplist (40 FR + 10 EN) + Expansion.*

* Values obtained using: qrels version dated 05/05/2025 and queries version dated 23/04/2025. Other values were obtained using: qrels and queries version dated 23/04/2024.

**Stemmers**:
Comparing the three stemming algorithms, Snowball consistently outperformed FrenchLight and FrenchMinimal, achieving the highest MAP and nDCG scores across both datasets.

**Tokenizers**:
Among the tokenizers, the Letter Tokenizer produced the best results, marginally outperforming Standard Tokenizer. Whitespace tokenizer, on the other hand, led to worse retrieval, likely due to its inability to handle punctuation and contractions present in the dataset.

**Analyzer Methods**:
We expanded our analysis by combining the best-performing Snowball stemmer with Letter and Standard Tokenizers and applying both stoplist filtering and query expansion strategies. The best performance resulted from employing the Letter tokenizer, tuned stoplist with 40 French + 10 English terms and abbreviation expansion. This indicates that stopword removal, combined with robust normalization and expansion, enhances precision and ranking quality.

**Other Methods**:
To enable a systematic and manageable evaluation, each method was tested independently under the simplifying assumption that they do not interact with each other. This approach allowed us to isolate and measure the individual contribution of each component:

- **Phrase** showed the best performance for 2022 data compared to other individual methods, achieving among the highest MAP and nDCG values in the category. This suggests that methods accommodating natural variations in language (and word order) is particularly effective in our setting.
  **Start**, on the other hand, yielded significant improvements over the baseline in 2023, but little to no gain on the 2022 data. The gains observed in 2023 suggest that this method became more effective with the newer dataset, contributing positively to retrieval quality.
- **Fuzzy** decidedly improved upon the baseline for both 2022 and 2023 datasets, suggesting that tolerance for minor typographical errors and spelling variations is highly valuable in this dataset. The increase in both evaluation metrics points to the method's ability to recover relevant documents that would otherwise be missed due to small discrepancies between query terms and document text.
- **Phrase + Fuzzy + Start**, the combination of these complementary strategies yielded the highest MAP and nDCG values for 2022 data. This result indicates that integrating contextual, positional, and fuzzy matching potentially creates a synergistic effect, enhancing overall system performance.
- **Highlights**, **N-grams** and **Start** showed inconsistent results, yielding marginally lower results than the baseline for 2022, but demonstrating a notable improvement on the 2023 data.
- **Dictionary**, **Synonyms**, **PRF**, **Word2vec**, and **LLM** only decreased the system performance compared to the baseline of *Snow + Letter + Stoplist (40FR+10EN) + Expansion*. This may be due to the addition of irrelevant or weakly related terms, which increased ambiguity or caused topic drift, ultimately reducing the system's precision.

As of May 5th, due to the API rate limit of 10 requests per minute, it was not feasible to evaluate the reranker within a reasonable timeframe. Processing approximately 7,000 queries would require over 15 hours, significantly hindering the ability to conduct timely and iterative experiments.

## 4.2. Test Results

The best-performing feature combination consists of using SnowballFilter as the stemmer, LetterTokenizer as the tokenizer, a stoplist comprising 40 French words and 10 English words, and abbreviation expansion. The core search strategy is based on the Phrase, Fuzzy, and Start features, which—when combined with Highlights and N-grams—do not improve the score further. All other features were found to degrade performance.

## 5. Conclusions and Future Work

Our systematic exploration of retrieval strategies for the CLEF LongEval task demonstrates that **traditional IR techniques**, when carefully tuned, form a robust foundation for modern search engines. The **Snowball stemmer** and **Letter tokenizer** emerged as the most effective normalization tools, with the combined pipeline `Snow + Letter + Stoplist (40FR+10EN) + Expansion` achieving strong baseline performance. Among query expansion methods, **phrase queries** (slop=5) and **fuzzy matching** (edit distance=2) consistently improved performance, while positional strategies like **Start** (weighting document beginnings) and **N-grams** delivered marginal gains on the 2023 dataset. Integrating complementary methods—such as `Phrase + Fuzzy + Start`—yielded the highest 2022 MAP (0.0979) and nDCG (0.1794), underscoring the value of hybrid approaches.

The data related to the 01/2023 dataset show more significant values, as they were tested using updated qrels files, whereas the 06/2022 data exhibit certain inconsistencies. Methods such as LLM-based expansion, pseudo-relevance feedback (PRF), as well as the use of dictionaries and synonyms, did not yield satisfactory results. Further fine-tuning of the associated parameters would be necessary to enhance their effectiveness and make their use beneficial for improving retrieval performance.

## 6. Group Members Contribution

**Alessandro Di Frenna** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Andrea Di Tillo** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Luca Pellegrini** Development of the parser, start and highlight features, construction of the stoplist.

**Marco Tomaiuoli** Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Artur Voit-Antal** Writing, formatting and assembling the report.

## References


[1] Longeval 2025: Clef longeval web retrieval train collection, https://clef-longeval.github.io/data/, 2024. Accessed: 2024-05-01.

[2] R. Alkhalifa, I. Bilal, H. Borkakoty, J. Camacho-Collados, R. Deveaud, A. El-Ebshihy, L. Espinosa-Anke, G. Gonzalez-Saez, P. Galuščáková, L. Goeuriot, E. Kochkina, M. Liakata, D. Loureiro, P. Mulhem, F. Piroi, M. Popel, C. Servan, H. T. Madabushi, A. Zubiaga, Overview of the clef-2023 longeval lab on longitudinal evaluation of model performance, in: A. Arampatzis, E. Kanoulas, T. Tsikrika, S. Vrochidis, A. Giachanou, D. Li, M. Aliannejadi, M. Vlachos, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of CLEF 2023, Lecture Notes in Computer Science, Springer Nature Switzerland, Cham, 2023, pp. 440–458.

[3] Cohere about, https://docs.cohere.com/reference/about, 2025. Accessed: 2025-05-01.

[4] Fuzzy query documentation by lucene, https://lucene.apache.org/core/9_10_0/core/org/apache/lucene/search/FuzzyQuery.html, 2024. Accessed: 2024-05-01.

[5] N. Lorenzon, vim-french-thesaurus, https://github.com/Nicoloren/vim-french-thesaurus, 2016. Accessed: 2024-05-01.

[6] Description - longeval 2025 train collection readme.pdf, https://researchdata.tuwien.ac.at/records/r643n-yc044, 2024. Accessed: 2025-05-01.

[7] S. E. Robertson, H. Zaragoza, The probabilistic relevance framework: BM25 and beyond, Foundations and Trends in Information Retrieval 3 (2009) 333–389.