SARD at LongEval: Longitudinal Evaluation of IR Systems by Using Query Rewriting and Hybrid Queries

Notebook for the LongEval Lab at CLEF 2025

Damiano Caon[1], Riccardo Dal Maschio[1], Alessandro Disarò[1], Sofia Maule[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This paper presents the participation of Team SARD in the *Conference and Labs of the Evaluation Forum (CLEF)* LongEval 2025 shared task, which investigates the longitudinal evaluation of information retrieval systems on evolving web collections. After a careful analysis on the dataset given to train the system, the group decided to first try some common techniques to improve performance and then focus on those that produced better experimental results. In particular, we explore a wide range of indexing and querying configurations by varying analyzers, language detection granularity, and query formulation strategies using the tools provided by the Apache Lucene framework. Additionally, we examine the impact of synonym-based query expansion using external lexical resources and query rewriting to correct typing errors. The evaluation is based on standard retrieval metrics: MAP, nDCG, and interpolated precision at standard recall levels, all computed using `trec_eval`. A custom MATLAB-based pipeline was developed to automate metric extraction, aggregation, and visualization across monthly snapshots. Our results indicate that document-level language detection and a combination of boolean and phrase queries improve performance in most scenarios. Conversely, synonym-based query expansion often degraded performance. In contrast, LLM-based query rewriting to fix user input errors led to noticeable improvements, demonstrating how even small interventions can enhance retrieval quality. We also observe high variability across queries and reflect on the implications for evaluation reliability. Future work will explore learning-to-rank approaches and the integration of large language models, contingent on the availability of higher-quality computational resources and adequate relevance judgments.

Keywords

Information Retrieval, LongEval, CLEF, Query Rewriting, Boolean Query, Phrase Query, Query Manipulation, trec_eval, MAP, nDCG, Interpolated precision at standard recall, Language Detection, Lucene, Retrieval Performance, Analyzer Comparison, Longitudinal Evaluation

1. Introduction

Information Retrieval (IR) systems are often trained on static datasets, but real-world data, especially on the Web, evolves continuously over time. As a result, their effectiveness tends to degrade when applied to new, unseen data. This issue is at the core of CLEF 2025 LongEval Task 1 (Web Retrieval)[1], which focuses on evaluating the robustness of IR models over time.

This lab provides an opportunity to evaluate retrieval systems on a temporally evolving collection with shared tasks and relevance judgments [2]. The goal is to investigate how systems behave in longitudinal settings, where both the data and the information needs change over time.

The aim of this report is to present our solution to this challenge. Our approach was to design an IR system capable of maintaining stable performance across temporal shifts. In particular, we focused on the development of custom language-specific analyzers for the preprocessing of documents and queries, and also the correction of user queries using `GPT-4 Turbo`. Unlike traditional query expansion methods, which add semantically related terms, our use of `GPT-4 Turbo` aims to rewrite and clarify ill-formed or noisy queries without altering their intent, resulting in more precise and effective matching with the indexed content.

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

damiano.caon@studenti.unipd.it (D. Caon); riccardo.dalmaschio@studenti.unipd.it (R. Dal Maschio); alessandro.disaro.1@studenti.unipd.it (A. Disarò); sofia.maule@studenti.unipd.it (S. Maule); nicola.ferro@unipd.it (N. Ferro)
https://www.dei.unipd.it/~ferro/ (N. Ferro)
0000-0001-9219-6239 (N. Ferro)

The paper is organized as follows: Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings; and Section 6 draws some conclusions and outlines directions for future work.

2. Related Work

To reduce the total time required to index all documents, we took inspiration from the work of a previous team participating in the LongEval challenge, specifically Team GWCA [3]. While their implementation leveraged multithreading for concurrent processing, we extended this idea by parallelizing not only the indexing tasks but also the document loading phase. In particular, we grouped input files into batches and assigned them to threads, rather than processing them one at a time.
Furthermore, to ground our methodology in prior experience, we consulted the LongEval notebook papers from the two previous years [4] [5]. These provided insight into the range of approaches attempted in earlier editions of the task and highlighted techniques that yielded competitive results.

3. Methodology

The methodology employed in this work follows the principles of *iterative design*. An initial baseline pipeline was defined at the beginning of the system development. As the work progressed, this structure was continuously refined through a cycle of implementation, testing, and improvement.
These iterations were not carried out blindly or based solely on trial and error; rather, they were guided by reasoned decisions aligned with established theoretical findings in the field of Information Retrieval. This approach enabled the progressive enhancement of the system based on both practical outcomes and theoretical grounding.

The architecture of the system, which also served as the foundational pipeline, was composed of the following key components:

- **Parser** – responsible for reading and structuring the raw input data;
- **Analyzer** – performed preprocessing tasks such as tokenization, stop-words removal, and stemming;
- **Indexer** – constructed and maintained the index used for information retrieval;
- **Searcher** – executed queries against the index and returned ranked results.

In order to manage the execution of the pipeline, a lightweight `Main` class was developed. This class was designed to maximize operational flexibility, allowing the user to control the system with the greatest degree of freedom possible. For instance, it is possible to select and process only a subset of months by modifying the initial month list, or to choose whether to perform indexing, searching, or both, by simply adjusting internal configuration flags. Additionally, users can easily set parameters for the search process, such as the number of documents to retrieve and the type of query to execute — either boolean or phrase queries. This design choice proved particularly valuable during system evaluation, as it avoided redundant indexing, a computationally expensive step, and enabled rapid testing of alternative configurations on previously built indexes.

The previously listed modules were implemented as standalone packages capable of interacting seamlessly with one another. Each component was designed with modularity in mind, allowing for flexible configuration and easy parameter adjustments. This architectural choice facilitated experimentation with different setups during the evaluation phase, supporting the iterative and theoretically-informed development process.

3.1. Parser

The Parser is responsible for reading the raw JSON files and transforming them into structured documents suitable for subsequent analysis and indexing. Although the component is relatively straightforward, particular care was taken to ensure robustness and efficiency during the design and implementation phase.
To process potentially very large datasets without running into memory limitations, the Parser was implemented using a streaming approach. Specifically, it employs a tokenized JSON reader (based on the Jackson library [6]) that reads the documents sequentially from the file. This design allowed the system to handle thousands of documents efficiently, without loading the entire dataset into memory.

During the parsing process, each document is cleaned to reduce textual noise and standardize the content before further processing. The following preprocessing steps are applied to the textual contents:

- Removal of emojis;
- Removal of URLs;
- Removal of HTML tags;
- Conversion to lowercase.

These cleaning operations were motivated by the need to simplify the input text and enhance the effectiveness of downstream components such as tokenization and indexing. In particular, removing highly variable or non-textual elements such as emojis, URLs, and HTML markup helped improve the consistency of the documents without sacrificing meaningful content.

Additionally, the Parser provides configurable options to selectively enable or disable the removal of emojis, URLs, and HTML tags. This feature was designed to offer flexibility during experimentation phases, even though in the final pipeline all cleaning steps were consistently applied.

To ensure robustness, the Parser incorporates a safety check that automatically skips invalid documents, defined as records missing either the identifier or the contents field. This prevented pipeline failures and minimized the impact of noise or inconsistencies in the raw data.

The documents produced by the Parser are instances of the `JsonDocument` class, a lightweight data model encapsulating the `id` and `contents` fields of each document. For consistency across modules, the field names (`id` and `contents`) are defined as constants within the `JsonDocument.FIELDS` inner class, ensuring compatibility with components such as the Analyzer and the Lucene-based indexing system.

Overall, the Parser contributed to establishing a clean and reliable document stream, serving as a solid foundation for the subsequent stages of the information retrieval pipeline.

3.2. Analyzer

To accommodate the multilingual nature of the dataset and support rapid experimentation, we developed a flexible analyzer architecture. The process began with an exploratory component (`TestAnalyzer`) and led to the definition of two final language-specific analyzers tailored to English and French documents.

3.2.1. TestAnalyzer

Initially, due to the presence of documents written in both English and French, we decided to build two separate analyzers—one for each language—so that the most appropriate processing could be applied based on the document's language. To support rapid experimentation with different configurations, we developed the `TestAnalyzer` class.

This component is designed to be flexible and fully configurable through external JSON files, which define all processing parameters without requiring any code modifications.
Each configuration file specifies the following parameters:

- `tokenizerType`: the type of tokenizer to use;
- `minLength - maxLength`: optional token length boundaries used to filter out tokens that are too short or too long. We tried using the range `0-100` to disable length filtering, and ranges such as `2-25` or `3-20` to restrict tokens to more meaningful lengths.
- `useEnglishPossessiveFilter`: whether to apply the English possessive filter;
- `stopListFilePath`: path to the custom stopword list file;
- `stemFilterType`: the type of stemming filter to apply;
- `language`: the language of the input text;

The `language` parameter also enables language-specific processing rules, such as the handling of contractions in French or possessive forms in English, and selects the most appropriate stemming algorithm.
The `TestAnalyzer` played a central role during the experimental phase, allowing us to evaluate various combinations of tokenization, stopword removal, and stemming for both English and French collections.

3.2.2. FrenchTextAnalyzer and EnglishTextAnalyzer

We selected the best-performing setup and formalized it into two dedicated analyzers: `FrenchTextAnalyzer` and `EnglishTextAnalyzer`. Each analyzer was designed to suit the linguistic characteristics of its respective language, while sharing a common processing pipeline structure.

- *Tokenization:* Both analyzers begin by breaking input text into tokens using Lucene's `StandardTokenizer`, which handles basic word segmentation.
- *Normalization:* A series of filters are then applied to standardize token formats:
  - `LowerCaseFilter` converts all characters to lowercase.
  - `ICUFoldingFilter` removes accents and diacritical marks to harmonize character encodings.
  - `PatternReplaceFilter` eliminates punctuation using a regular expression.
  - `LengthFilter` retains only tokens between 2 and 20 characters, removing overly short or long terms.
- *Stopword Removal:* Language-specific stopword lists are applied using `StopFilter`. We used two separate lists: `french_stopwords.txt` and `english_stopwords.txt`, both taken from Oracle's default stoplists[1].
- *Stemming and Language-Specific Filters:*
  - The `FrenchTextAnalyzer` includes an `ElisionFilter` to remove contractions like "l'", "d'", and "c'", which are common in French. For stemming, it uses `FrenchMinimalStemFilter`, which we found to perform best by balancing morphological reduction with semantic retention. We also tested `FrenchLightStemFilter` and the `SnowballFilter` with `FrenchStemmer`, but they were less effective in preserving useful lexical distinctions.
  - The `EnglishTextAnalyzer` applies an `EnglishPossessiveFilter` to strip possessive suffixes (e.g., "'s"), reducing noise from possessive forms. It then uses `EnglishMinimalStemFilter`, which outperformed alternatives like `PorterStemFilter` and the `SnowballFilter` with `EnglishStemmer` by avoiding overstemming and maintaining semantic clarity.

Overall, this dual-analyzer setup allowed us to tailor preprocessing to the specific needs of each language, improving indexing consistency and retrieval effectiveness.

[1] https://docs.oracle.com/en/database/oracle/oracle-database/21/ccref/oracle-text-supplied-stoplists.html#GUID-5DAF7499-5EBA-41E6-AF1A-C3BD2C08F88F

3.3. Indexer

The Indexer is responsible for creating the index used for document retrieval. In the final versions of our systems, each `Lucene Document` in the index contains five fields: the document ID, the content (if in English), the content (if in French), the language of the document, and the name of the JSON file from the collection where the original document is located. Having two separate content fields allowed us to apply different analyzers to each, effectively handling both languages. Although it is a fairly standard component, with limited impact on the overall effectiveness of the system, we focused on improving its performance to facilitate the system testing phase. In particular, we concentrated our efforts on two key aspects: multithreading and efficient language detection.

To reduce the total time required to index all documents, and inspired by the work of a previous team participating in the LongEval challenge [3], we designed the Indexer to leverage multiple processors using a multithreaded approach. This optimization resulted in a 50% reduction in indexing time compared to the single-threaded implementation.

Since the Indexer is also responsible for selecting the appropriate Analyzer based on the document's language, we tried to make the language detection process both fast and reliable. Initially, we considered using the Lingua library [7], a widely adopted Java tool for language detection, but we observed that it introduced a significant time overhead. Consequently, we implemented a custom language detection class using a hybrid approach, analyzing only the beginning of each document to minimize processing time. The first version of this class was developed to recognize the language of a sentence, but then it was modified to work with full documents. The steps of our language detection process are as follows:

- If the document contains enough words, we use two predefined stopword lists (one for English and one for French) to estimate the language, counting the occurrences of stopwords from each list;
- If the resulting confidence score, i.e. the difference in the counts divided by the total number of stopword occurrences, exceeds a predefined threshold, the detected language is returned;
- If confidence is too low or the text is too short, we fall back to the Lingua library, restricting detection to English and French;
- If the result is UNKNOWN (e.g., if the text is in a different language), we default to French.

After experimenting with various parameter settings, we selected the following configuration:

- A minimum of 10 words to attempt detection using the stopword approach;
- A maximum of 500 words to process per document;
- A confidence threshold of 0.2 for the stopword-based detection.

This configuration was tested on approximately 46000 documents, corresponding to the first three JSON files from the September 2022 snapshot. The results showed that in roughly 99% of the cases, the language detected by our hybrid method matched the one identified by Lingua when applied to the full text. Furthermore, a manual review of the mismatches revealed that in almost 60% of the cases, the discrepancy was not actually a misclassification, as the texts were multilingual or written in a different language altogether. The manual review also helped us better understand common sources of detection inconsistency, such as:

- Mixed French and English texts (e.g., `doc141`);
- Lists from e-commerce sites or product catalogs (e.g., `doc1354517`);
- Documents written in various languages and sometimes different alphabets (e.g., `doc258`);
- Cookie banners or browser-related boilerplate (e.g., `doc3760`).

Despite a slight loss in accuracy compared to the full-text Lingua approach, our hybrid method achieved a 95.6% speedup in language detection, leading to a significant overall reduction in indexing time. It is

worth mentioning that the language detection class also contains a method to recognize the language of single words by checking two large vocabularies (one for French and one for English), with a fallback on Lingua if the term is not found there. However, this method proved unreliable, often failing with complex or uncommon terms, and was therefore not used in the final systems.

Finally, by inspecting the collection, we noticed that some documents were repeated multiple times within the same snapshot (e.g., `doc3173607` appears 56 times in the September 2022 snapshot). To address this, the Indexer was modified to check whether each document had already been processed before indexing it.

3.4. Searcher

The `Searcher` component is responsible for handling the search functionality in the information retrieval system. It is designed to query the index, process the results, and output the relevant documents based on the provided queries and QRELS. The implementation supports various features and settings to try different approaches and improve or comparing retrieval processes, such as query expansion, handling multiple languages (English and French), and parallelization to improve (time) performance. The configurability of the Searcher object has allowed us to easily swap from a setup to the other, in order to try as many configurations as possible, using different types of analyzers with different query types.

3.4.1. Query Processing

The search process begins by reading the set of provided topics: a custom `QueryReader` object is used by the `Searcher` class to parse the queries of a certain month and converting them in `QualityQuery` objects, for mapping reasons, which can easily be processed by the system. For each of these objects, a query based on the provided content, either in the form of a Lucene's `BooleanQuery` or a `PhraseQuery`, depending on the configuration, is created. The `BooleanQuery` is particularly useful for basic matching, where the query terms are connected using logical operators, such as `OR`. On the other hand, `PhraseQuery` is employed when the order of the terms in the query is important, allowing for a better matching of phrases rather than individual terms. The `Searcher` object then also accepts two analyzers, both for French and English, which hopefully have to be the same ones used to index the corpora, to guarantee the best match based on the detected language of query terms. Language detection, though, proved to be a challenging factor that impacted the search system's performance. Different systems exhibited inconsistent results depending on the configurations used. Even though we developed a custom `FrenchEnglishDetector`, with better performances in terms of time, the Lingua [7] `LanguageDetector` seemed to perform better in average, even if it was slower.

3.4.2. Parallelization

To enhance the efficiency of the search process, especially to handle the large indexes and a big number of queries, the Searcher has been designed to utilize parallel processing. The search tasks for each topic are executed in parallel using Java's `ExecutorService`. This approach enables the system to process multiple queries simultaneously, reducing the overall search time and improving scalability. The number of threads used for parallelization is configurable, with a default that matches the number of available processors on the machine. This ensures optimal resource utilization and allows for fine-tuning based on system specifications.
To have and indicative measure of how much the performance improved, the searching was performed roughly 70% quicker with 4 threads instead of 1.

3.4.3. Query Expansion

Another challenging aspect has been query expansion. The basic process was to generate synonyms from the query terms and analyze them to create the stems to build the queries to perform on the index. This way we hoped to increase the potential matches for every query. To better understand the query expansion methods used, see Section 3.4.5.
For the `BooleanQuery` object the process was simply adding synonyms to them (with the OR clause). However, we immediately noted a drastic fall in performance even when adding just a handful of synonyms to the queries. We therefore decided to reduce that number as much as possible and to boost, or more accurately, to penalize the strings not derived directly from the query terms, using Lucene's `BoostQuery` object. It is important to note that even boosting (or not boosting) a single term with a factor of 1 worsened performance.
Regarding the `PhraseQuery` objects, a simple way of enhancing queries has been to use the slop parameter to generate permutations of the starting query. In addition, to better manage more complex query expansion with synonyms, we decided to use a Lucene object designed specifically for this application, the `MultiPhraseQuery` class.
Unfortunately, as can be seen in Section 5, these procedures did not help improve performance.

3.4.4. GPT elaborated queries

To improve the quality of the queries, we processed them using `GPT-4.0 Turbo` asking the *Large Language Model (LLM)* to restore proper diacritics, correct misspellings, and rewrite overly long queries. In a real IR system, this processing would typically be done at runtime each time a new query is submitted by a user. However, since we needed to handle a large number of queries, we chose to process them in advance to avoid repeating the same procedure at every run (also considering that the APIs for the chosen LLM are not free). Since the snapshots from June 2022 to September 2022 contained numerous queries related to adult content, which caused problems when processed by GPT, we decided to collect all queries from the remaining training set, remove duplicates, and improve only those. As a result, queries from October 2022 to February 2023 were fully rewritten, while for the earlier months the proportion of rewritten queries ranged from 78% to 88%.

3.4.5. Synonyms Generator

To perform query expansion, we developed a class capable of returning a list of synonyms for a given English or French word. Our original idea was to accomplish this using two different techniques: one based on fixed dictionaries and the other leveraging LLMs.

For the dictionary-based approach, we used WordNet [8] for English queries and WOLF [9], its French counterpart, for French queries. While we had a CSV version of WordNet readily available, we had to pre-process the WOLF XML file to extract synonym lists for each word. At runtime, the extracted synonym sets are parsed into dictionaries, and when the proper methods are called, the `SynonymsGenerator` provides a list of synonyms for the given word. This approach is very fast, but the generated synonyms are not selected with regard to the context of the query, often leading to query drift and poor effectiveness. Additionally, there were some strange links in the dictionaries, resulting, for example, in `gondola` being provided as a synonym for the English term `car`, or `clientélisme` for the French term `voiture`.
To improve the quality of the synonyms, we first attempted to build a local Python server using Flask and the model `LLaMA-3.2-1B-Instruct`, but this approach proved unsuccessful. Despite carefully designed prompts explicitly instructing the model to return only the expanded query, the output was often inconsistent or poorly formatted. For instance, when expanding the French query:

```
évolution voiture
```

the model returned:

```
évolution de la voiture, évolution automobile, évolution du véhicule,
évolution des voitures, avancement technique, amélioration continue,
innovation technologique, avancée scientifique, progrès industriel,
développement numérique, amélioration continue de la voiture, avancement
technologique, progrès de l'automobile, évolution de l'industrie
```

which was satisfactory in terms of format but slightly too broad in terms of relevance. However, for the English version of the same query:

```
car evolution
```

the model produced the following output:

```
Original Query:\n Car evolution refers to the process of changing a
vehicle's design, features, and technologies over time.\n Synonyms/variants:\n
1. Automotive evolution\n 2. Vehicle development\n 3. Car transformation\n
4. Motor vehicle evolution\n 5. Vehicle modernization
```

despite the prompt being identical in structure and clearly requesting a clean list of synonyms in plain text, without any preamble or line breaks. Furthermore, the server required nearly 15 seconds to process each request, making it impractical for handling the large number of queries provided in the LongEval challenge. Similar problems were encountered when trying `GPT-3.5 Turbo`, whereas they disappeared when testing with `GPT-4.0 Turbo`. Unfortunately, the cost of using the latter model's APIs was prohibitive given the volume of queries to be processed, and the use of LLMs for synonym generation was ultimately abandoned.

3.5. Experimental Strategy

Our experimental process was structured in multiple stages to explore the impact of different indexing and querying configurations on retrieval performance. The training set results that guided our methodological choices are presented in Section 5.2.

Given the multilingual nature of the documents, we initially performed runs using sentence-level language detection, where each sentence in a document was analyzed individually to assign language-specific analyzers. This approach ensured that documents containing both French and English sentences (often due to browser-related boilerplate) were processed with the appropriate analyzers for each part. In the first phase, we tested all available analyzer configurations using `BooleanQuery` in the searcher, fixing the maximum number of documents retrieved per topic at 300, to identify the most effective configuration. The results showed that the best analyzer was `minAnaBool1`, so all subsequent steps were carried out using this configuration.

Next, we aimed to increase indexing speed by switching to document-level language detection, where the entire document was assigned to either the English or French analyzer based on a global analysis, instead of detecting the language at the sentence-level (based on a sample of 1000 documents, the average number of sentences per document was approximately 30). As expected, this change reduced indexing time and, surprisingly, also improved retrieval performance.

To enhance precision, we tested the same analyzer using `PhraseQuery` to prioritize retrieving documents where query terms appeared close to each other, allowing for a small margin to avoid overly strict matching. A comparison between the previous `BooleanQuery` run and the new `PhraseQuery` run confirmed the increase in effectiveness. We also increased the maximum number of retrievable documents to 1000 for both `BooleanQuery` and `PhraseQuery`, observing only a negligible gain in effectiveness but a notable increase in the number of relevant documents retrieved with `BooleanQuery` (an average of 32%). As a result, we decided to set the maximum retrievable documents to 1000 for

`BooleanQuery`, while keeping it at 300 for `PhraseQuery`, since the increase in search time was not justified by the marginal gain in recall.

Recognizing that Qwant users often omit diacritics and that some queries contained errors or were poorly structured, we applied LLM-based query correction as described in Section 3.4.4. The results showed that queries re-elaborated by `GPT-4.0 Turbo` improved performance for `BooleanQuery` but slightly decreased it for `PhraseQuery`.

To further increase recall, we tested query expansion as described in Section 3.4.5, but this approach backfired: not only did we retrieve fewer relevant documents, but we also observed a drop in performance compared to the baseline, particularly for `PhraseQuery`. This was likely due to the poor quality of the `SynonymsGenerator`, which led to query drift. Analyzing the topic-level MAP from the December 2022 run, we observed that:

- 4227 queries (55%) were identical regardless of the introduction of query expansion;
- 1221 queries (16%) improved, with an average increase of 0.0542;
- 2213 queries (29%) worsened, with an average decrease of 0.0399.

This indicates that while some generated synonyms were useful, most were not. Note also that we used the GPT-elaborated queries for `PhraseQuery` as well, due to the structure of the `SynonymsGenerator`, which favors words with proper diacritics.

Finally, we sought to combine the strong precision of `PhraseQuery` with the broader recall of `BooleanQuery`, performing both approaches together and applying different boost values when combining them. We tested combinations both with and without GPT-elaborated queries, based on what we had learned about their respective effects. The best results were achieved using boost values of 1.5 for `PhraseQuery` and 0.9 for `BooleanQuery`, with GPT-elaborated queries. This combination yielded the highest MAP, nDCG, and number of relevant documents retrieved.

4. Experimental Setup

4.1. Collection

The experiments were conducted using the LongEval-Web Retrieval collection[2], a benchmark specifically designed to study the robustness and stability of web search engines over time. The collection aims to address two fundamental questions in IR:

- How does a search engine behave as the underlying collection of documents evolves?
- When and how frequently should an IR system be updated to maintain performance as the document collection changes?

The queries were created in French and extracted from real user logs, while the documents may contain content in multiple languages, primarily French.
The dataset is organized into 15 monthly snapshots (June 2022 – August 2023), which enables fine-grained evaluation of IR systems over time. Compared to previous LongEval editions (2023 and 2024), the 2025 iteration of the collection features an expanded set of snapshots, allowing for more detailed analysis of collection dynamics.

The dataset is composed of:

- **Training set**: nine monthly snapshots from June 2022 to February 2023, containing approximately 18 million documents, more than 13,000 queries, and associated relevance judgments (qrels);

[2] https://clef-longeval.github.io/data/

- **Test set**: six monthly snapshots from March 2023 to August 2023, including documents, queries and relative qrels;
- **Relevance judgments** were derived from implicit user feedback, collected automatically via a click model based on Dynamic Bayesian Networks trained on Qwant search logs. The model outputs a probability of document attractiveness, which is then mapped to a 3-point relevance scale (0 = not relevant, 1 = relevant, 2 = highly relevant). This automatic estimation process is scheduled to be complemented with explicit human relevance assessments after the official submission deadline.

Overall, the full collection comprises fifteen distinct datasets, each corresponding to a specific monthly snapshot.

During the development of this project, we observed a significant performance gap between the first four months of the training collection and the subsequent five. To better understand the reasons behind this, and with the hope of finding ways to improve our systems' scores, we spent considerable time analyzing the provided documents, queries and qrels. We wrote some Python scripts to speed up frequent tasks, such as retrieving a document by its ID, and manually examined the files to identify potential issues. We discovered that, at least for some queries, there is a clear mismatch between the queries file and the qrels file. For example, query ID 1 for June 2022 refers to an adult content website, but the documents marked as highly relevant in the qrels were about World War I (which actually corresponds to query 3). A similar situation occurred with query 27, `adenovirus`, which was associated in the qrels with a document about energy efficiency. Moreover, although the LongEval challenge website states that filters were applied to exclude adult content from the collection, it was not the case for the first months, possibly explaining the cause of these mismatches. Further analysis revealed that even the last five months of the training collection were not entirely immune to this problem, although the impact there was smaller. In any case, this manual review was crucial in helping us decide which approaches to pursue and which to abandon as using certain methods would have been misleading without reliable qrels to serve as ground truth.
To address the issue, the organizers released an updated version of the training queries and qrels on May 5th, 2025. While the misalignment between queries and qrels was resolved, the qrels still included lines referring to queries that were not provided or to topics with no associated relevant documents.

4.2. Measures

To assess the effectiveness of our retrieval system, we adopted three widely used evaluation metrics: *Mean Average Precision (MAP)*, *Normalized Discounted Cumulated Gain (nDCG)*, and interpolated precision at standard recall levels (iPrec@Recall). These metrics were selected because they capture complementary aspects of retrieval quality:

- MAP measures the overall precision by averaging the precision scores at the ranks where relevant documents occur. It reflects how well the system retrieves all relevant documents.
- nDCG evaluates the quality of the ranking by assigning higher importance to relevant documents that appear earlier in the results list, which is especially important in web search scenarios.
- Interpolated precision at standard recall levels (iPrec@Recall) provides a global view of system performance by reporting the highest precision achieved for any recall level greater than or equal to a given threshold. This metric is a well-established tool for analyzing the trade-off between precision and recall in a smooth and interpretable way.

The evaluation was performed using `trec_eval`[3], a standard tool for information retrieval benchmarking. We used it to compare our system's output (run files) against the official relevance judgments (qrels) provided by the LongEval organizers [1]. This ensured that our results followed a consistent and widely accepted evaluation protocol.

[3] `trec_eval` repository: https://github.com/usnistgov/trec_eval

4.2.1. MATLAB

To support the analysis of our retrieval experiments, we developed a modular MATLAB framework designed to transform the output of `trec_eval` into structured matrices suitable for both manipulation and visualization of data. The framework handles monthly evaluation files for each run and metric, converting them into consistent matrices depending on the specific measure.
These matrices are then used in two main ways: to produce clear comparative plots of system performance over time, and to enable statistical testing via methods such as *ANalysis Of VAriance (ANOVA)*. This dual purpose made the framework essential to our workflow, allowing us to efficiently organize, interpret, and compare the results of a wide range of system configurations over various time-snapshots. The following sections describe each script and its specific role in the evaluation process[4].

Matrix Construction with Query IDs

This script processes the output files generated by `trec_eval` and converts them into structured MATLAB matrices that explicitly preserve query identifiers. It supports three standard evaluation metrics: MAP, nDCG, and interpolated precision at standard recall levels (iPrec@recall). The user can choose to elaborate a single metric or all three sequentially.
For MAP and nDCG, the script constructs a matrix where each row corresponds to a query, the first column stores the query ID, and the remaining columns contain the scores for each month. This format enables direct traceability and is suitable for the following analyses that require maintaining the link between scores and their originating queries. For iPrec@recall, which aggregates scores across queries, the matrix has fixed recall levels as rows and months as columns.
The script automatically detects the unique set of query IDs present across all months, aligns scores accordingly, and fills missing values with `NaN` to maintain dimensional consistency. The resulting matrices are saved in dedicated `.mat` files following a standardized naming convention. This structure is designed to speed up the process of importing in MATLAB the huge number of `trec_eval` measurements for all our runs.

Cross-System Comparison with Confidence Intervals

This script performs comparative evaluation of multiple retrieval systems over time, using mean scores and 95% confidence intervals computed via Student's t-distribution. The processing is based on pre-aligned matrices generated by the previously described script.
The script begins by aligning the set of query IDs across all systems. For each system, it inserts `NaN` rows for queries that were retrieved by at least one other system but are missing in the current one. Missing values are then zero-filled, but only in time snapshots where at least one system reported valid data. This strategy avoids incorrectly assigning a score of zero to a query that may simply have been absent from a particular month's snapshot, such cases are better represented as `NaN` to preserve correctness in downstream processing. Finally, rows containing no informative values across any system are removed as a safeguard, although such cases are not expected in practice. Once alignment is complete, query IDs are stripped and overall means, per system and metric, are printed (these means are reported in Table 4).
For each metric, the script can generate two types of plots: one showing mean performance over time, and another using grouped error bars to visualize confidence intervals which are calculated for each system and time point. Systems are automatically sorted by overall average performance for easier comparison. A similar visualization is also supported for iPrec@Recall.

Month-Specific ANOVA and Boxplot Analysis

This script performs a fine-grained statistical analysis of multiple retrieval systems by focusing on a single time snapshot. It operates on per-query matrices generated for MAP and nDCG, evaluating

[4]It is important to clarify that this section is meant to illustrate the capabilities of the described scripts. As such, we do not present every possible plot that could be generated, but rather focus on those that are most informative and relevant to our analysis.

system performance at a selected month using two-way ANOVA and visual comparison.
The preprocessing and alignment of query scores across systems follow the same procedure described in the previous script. After alignment, the script isolates the column corresponding to the target month and constructs an evaluation matrix for each metric (where rows corresponds to queries and columns to systems), which will then be fed to MATLAB's `anova2` function.
For both MAP and nDCG, the script runs a two-way ANOVA to assess whether performance differences between systems are statistically significant. The results are visualized using MATLAB's `multcompare` tool and sorted boxplots that illustrate distributional differences in scores across systems.
This script provides a more focused comparison than time-series analysis, enabling robust identification of statistical significant performance differences at specific stages of the evaluation timeline.

4.3. Repository

The full source code of the project is available at the following Bitbucket repository:
https://bitbucket.org/upd-dei-stud-prj/seupd2425-sard/src/master/
The repository contains all scripts and resources needed to reproduce the indexing, retrieval, and evaluation process described in this report.

4.4. Hardware

To perform the experimentations, multiple hardware setups were used. Due to the high computational cost and long execution time of the indexing process, especially when testing different analyzers, the group decided to distribute the workload among its members. Each member was responsible for indexing the dataset using a specific analyzer on their personal machine. To maintain consistency, each member also conducted the corresponding search experiments on the index they had created, using a searcher configured with the same analyzer setup. This ensured the validity of the results while balancing both the indexing and searching load across the group. The table below summarizes the hardware specifications of the systems used:

Table 1
Hardware Specifications Used for Indexing and Searching

Member	Operating System	CPU	RAM	Storage
Damiano Caon	Windows 11	i5-1145G7	16 GB	SSD 1 TB
Riccardo Dal Maschio	Windows 11	i7 8th gen	16 GB	SSD 512 GB
Alessandro Disarò	Windows 10.0.19045	i7 10th gen, 4 core	16 GB	SSD 512 GB
Sofia Maule	Windows 11	i5-1035G1	8 GB	SSD 256 GB

5. Results and Discussion

5.1. Runs Description

In this section, we provide a brief description of the runs executed using the updated queries and qrels released after the correction of the originally distributed files[5] produced (runs used just for parameter tuning and with previous queries and qrels are omitted), along with the corresponding run identifiers used later in the performance evaluation.
All runs were executed using the `BM25Similarity` scoring function, they rely on the French and English documents collection, with filters and preprocessing steps as described in Section 3.2, and use

[5]The suffix `_newQuery` in the run names is used to distinguish them from those based on the old queries and qrels. The corresponding evaluation scores for the old runs are still available in our project repository for comparison purposes.

stoplists `english_stopwords.txt` and `french_stopwords.txt`, unless otherwise specified.

All initial runs employed a `BooleanQuery` strategy during the search phase, aimed at evaluating the effectiveness of different analyzers:

- **stdAnaBool_newQuery**: uses Lucene's `StandardAnalyzer` for both English and French collections;
- **minAnaBool1_newQuery**: uses `FrenchMinimalStemFilter` and `EnglishMinimalStemFilter` for stemming;
- **liminAnaBool_newQuery**: combines `FrenchLightStemFilter` stemming for French text and `EnglishMinimalStemFilter` stemming for English text;
- **snowAnaBool_newQuery**: applies `SnowballFilter` stemming for both languages;
- **porterAna1Bool_newQuery**: applies `FrenchLightStemFilter` for French and `PorterStemFilter` for English.

Once the best analyzer configuration was identified as the one used in `minAnaBool1_newQuery`, we retained it for all subsequent runs. Additionally, all following runs used the index built with document-level language detection, varying the query types, the number of retrievable documents, and the applied query manipulations:

- **minAnaBool300_newQuery**: uses Boolean queries, retrieves at most 300 documents per topic;
- **minAnaBool1000_newQuery**: uses Boolean queries, retrieves at most 1000 documents per topic;
- **minAnaPhr300_newQuery**: uses Phrase queries, retrieves at most 300 documents per topic;
- **minAnaBool1000_GPT_newQuery**: uses Boolean queries, retrieves at most 1000 documents per topic, with GPT-processed queries;
- **minAnaPhr300_GPT_newQuery**: uses Phrase queries, retrieves at most 300 documents per topic, with GPT-processed queries;
- **minAnaBool1000_GPT_QEDict_newQuery**: uses Boolean queries, retrieves at most 1000 documents per topic, with GPT-processed queries, dictionary-based query expansion;
- **minAnaPhr300_GPT_QEDict_newQuery**: uses Phrase queries, retrieves at most 300 documents per topic, with GPT-processed queries, dictionary-based query expansion;
- **minAnaMixed_newQuery**: uses combination of Boolean (boost 0.9) and Phrase (boost 2.0) queries, retrieves at most 1000 documents per topic (best model without GPT-processed queries);
- **minAnaMixed3_GPT_newQuery**: uses combination of Boolean (boost 0.9) and Phrase (boost 1.5) queries, retrieves at most 1000 documents per topic, with GPT-processed queries (best model).

5.2. Results on the Training Set

This section presents a comparative evaluation of system performance on the training set across configurations and query strategies. The metrics considered are MAP, nDCG, and interpolated precision at standard recall levels (`iPrec@Recall`), as introduced in Section 4.2. While all reported metrics were taken into account in guiding the development process, we present here only the plots of the metrics that, in each case, best illustrate the results, to improve clarity. All visualizations were generated using the MATLAB framework described in Section 4.2.1.
It should be noted that the key decisions were originally based on the old queries and qrels, but were later confirmed by the results obtained with the updated ones. These are the results presented in this section. As mentioned in Section 3.5, in the early stage of experimentation, our goal was to determine the most suitable analyzer to adopt in the subsequent configurations. By comparing five different analyzers using Boolean queries, we observed that all analyzers with different stemming behaved similarly in terms of MAP and nDCG. The only configuration that consistently underperforms across all metrics is `stdAnaBool_newQuery`, based on Lucene's default `StandardAnalyzer`. Among the

remaining configurations, `minAnaBool1_newQuery`, based on minimal stemming, shows slightly higher performance, as can be clearly observed in the interpolated precision plot (Figure 1).
Based on these results, we adopted its configuration as the reference analyzer in all subsequent experiments.

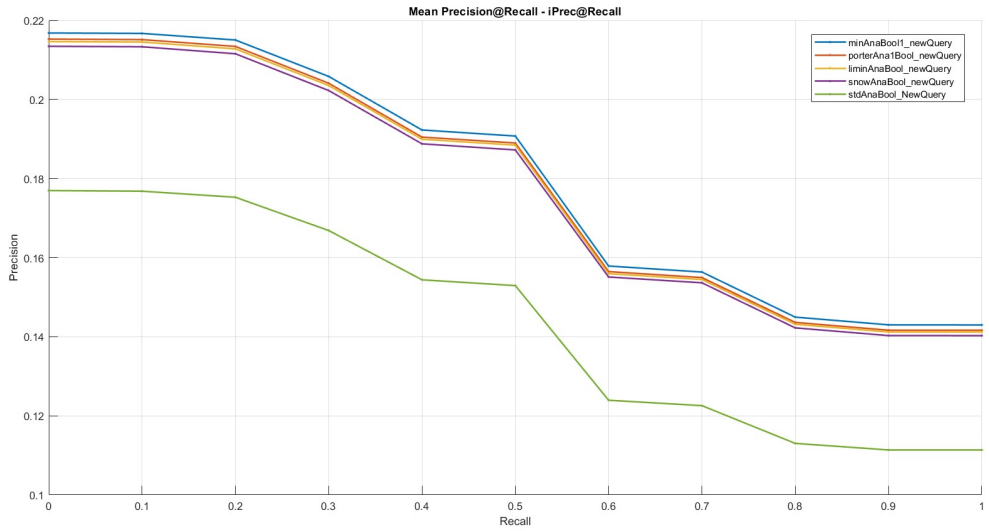


Figure 1: Mean interpolated precision at recall levels for Boolean systems.

After selecting the best analyzer, we proceeded to test additional configurations involving different query strategies, document limits, and query reformulations. The figures below report the comparative results of these extended experiments for MAP (Figure 2) and nDCG (Figure 3).

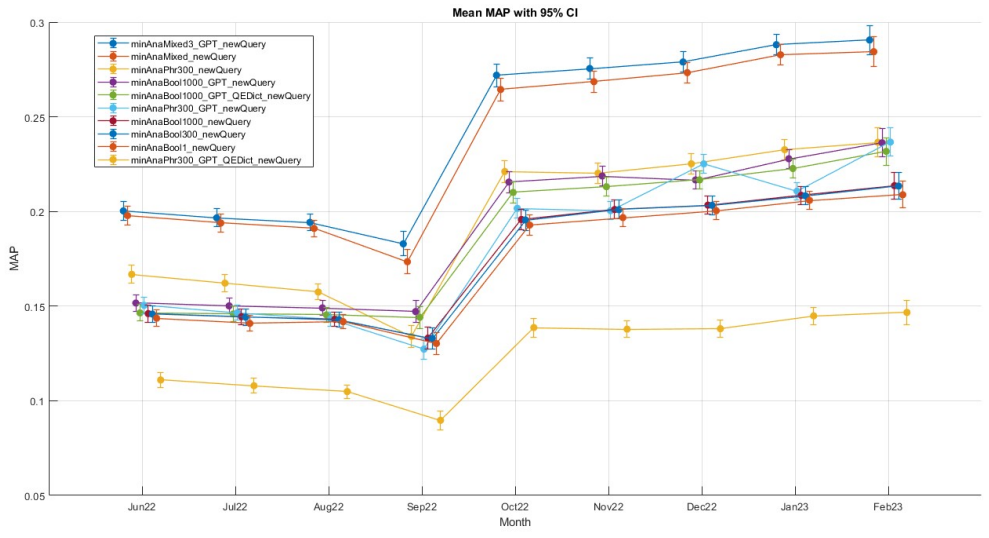


Figure 2: Mean MAP over months with 95% C.I.

The results showed that: increasing the number of retrieved documents was effective for Boolean queries; query rewriting using `GPT-4.0 Turbo` proved beneficial for Boolean queries but led to a drop

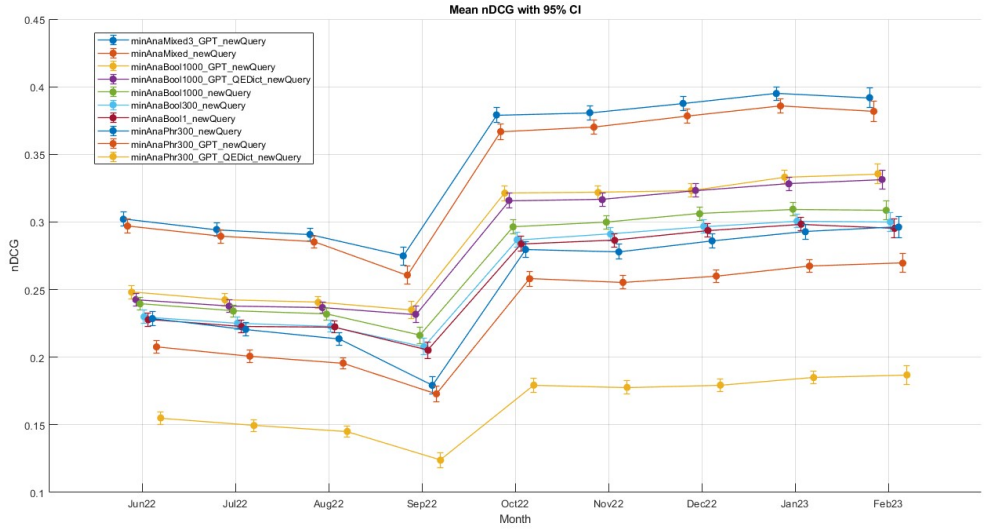


Figure 3: Mean nDCG over months with 95% C.I.

in scores for Phrase queries; query expansion slightly decreased the performance of Boolean queries and significantly worsened the results for Phrase queries; the hybrid strategy combining Boolean and Phrase queries turned out to be the most effective overall. It is also interesting to note that Phrase queries led to an improvement in MAP, while causing a significant drop in nDCG, indicating that the system is effective at identifying relevant documents, but less so at ranking them appropriately.
Table 2 shows MAP and nDCG scores for all the systems discussed; the ones submitted to the LongEval challenge are shown in bold.

Table 2
Performance of the runs

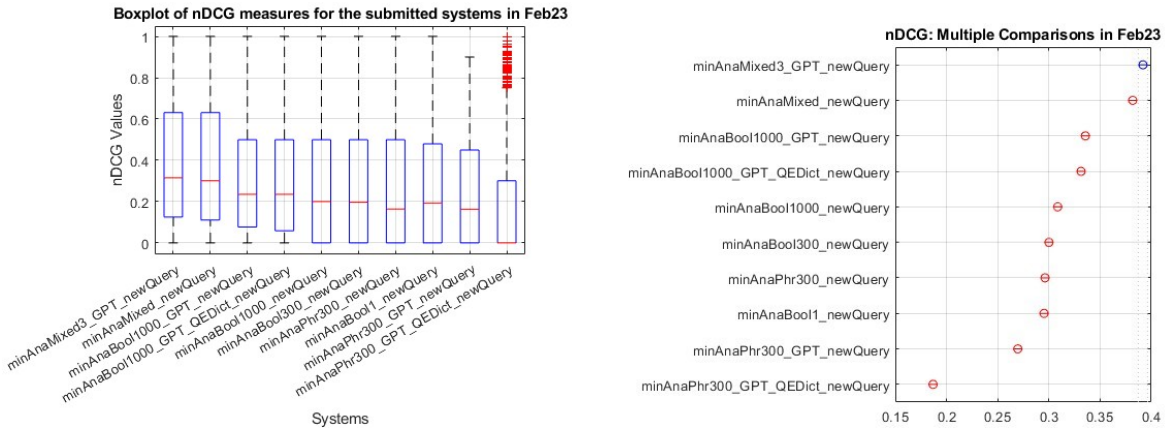
Run	MAP	nDCG
minAnaMixed3_GPT_newQuery	0.2421	0.3441
minAnaMixed_newQuery	0.2366	0.3350
minAnaPhr300_newQuery	0.1951	0.2527
minAnaBool1000_GPT_newQuery	0.1903	0.2891
minAnaPhr300_GPT_newQuery	0.1825	0.2319
minAnaBool1000_GPT_QEDict_newQuery	0.1862	0.2849
minAnaBool1000_newQuery	0.1765	0.2713
minAnaBool300_newQuery	0.1763	0.2623
minAnaBool1_newQuery	0.1735	0.2595
porterAna1Bool_newQuery	0.1721	0.2586
liminAnaBool_newQuery	0.1716	0.2579
snowAnaBool_newQuery	0.1704	0.2562
stdAnaBool_newQuery	0.1383	0.2144
minAnaPhr300_GPT_QEDict_newQuery	0.1244	0.1646

To improve the interpretability of our findings, we paired the raw performance results with statistical testing. Specifically, we performed a two-way ANOVA followed by Tukey's HSD multiple comparisons on the results for the 2023-02 snapshot, focusing on nDCG as the main evaluation metric. This particular snapshot was selected due to its high similarity to those included in the test collection [10]. The results of this analysis are reported in Table 3 and Figure 4b, while the corresponding boxplots (Figure 4a) highlight the distribution and relative stability of each system's performance. All statistical tests were

conducted at a significance level $\alpha = 0.05$.

Table 3
Two-way ANOVA table for nDCG scores on training runs (Feb 23)

Source	SS	df	MS	F	Prob > F
Columns	116.8249	9	12.9805	369.9507	0
Rows	6.2050e+03	7949	0.7806	22.2475	0
Error	2.5102e+03	71541	0.0351	–	–
Total	8.8320e+03	79499	–	–	–



(a) Boxplot of nDCG scores

(b) Multiple comparisons of nDCG scores

Figure 4: nDCG analysis on training runs for Feb 23.

The results clearly indicate that the systems are not all statistically equivalent; in particular, the submitted ones are all significantly different from one another. The system based on Phrase queries combined with query expansion shows a distinctive pattern in the boxplots: due to the presence of many topics with zero scores, the few with high scores appear as outliers in the overall distribution.

5.3. Results on the Test Set

In this section, we report the results obtained by running the selected configurations on the official LongEval test set. Only the systems marked in bold in Table 2 (training results) were submitted, and are thus included here. These systems reflect the most effective combinations of query strategies, analyzers, and reformulation techniques identified during development. We have also included `minAnaBool300_testColl` as a baseline to assess how our developments have impacted performance, as it was the first system built after fixing the indexing strategy.
The evaluation metrics adopted are consistent with those used for the training set. However, due to the qrels issues highlighted in Section 4.1, the numerical results cannot be directly compared between the training and test sets.

To show the performance trends across the different test months, we include two plots: Figure 5 for MAP and Figure 6 for nDCG, while the corresponding average scores are reported in Table 4. The results confirm the trends observed on the training set in terms of the relative ranking of systems, and highlight how the last snapshot, which is the most distant in time from the training set, shows a degradation in both metrics.

A more detailed analysis, involving two-way ANOVA and Tukey's HSD multiple comparisons, was conducted on a subset of months from the test collection. In particular, we focused on March, April, and August 2023, which correspond to the first, second, and last snapshots in the collection, respectively.

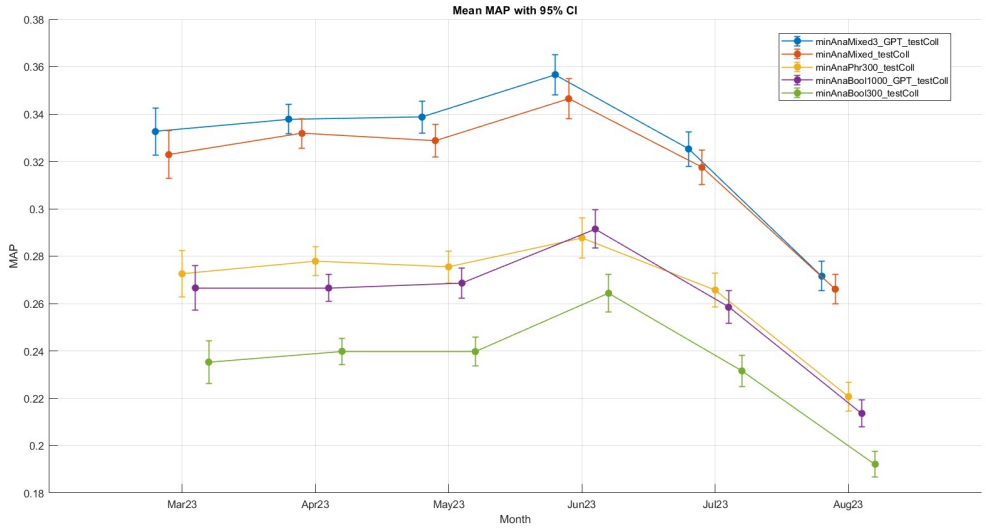


Figure 5: Mean MAP over months with 95% C.I.

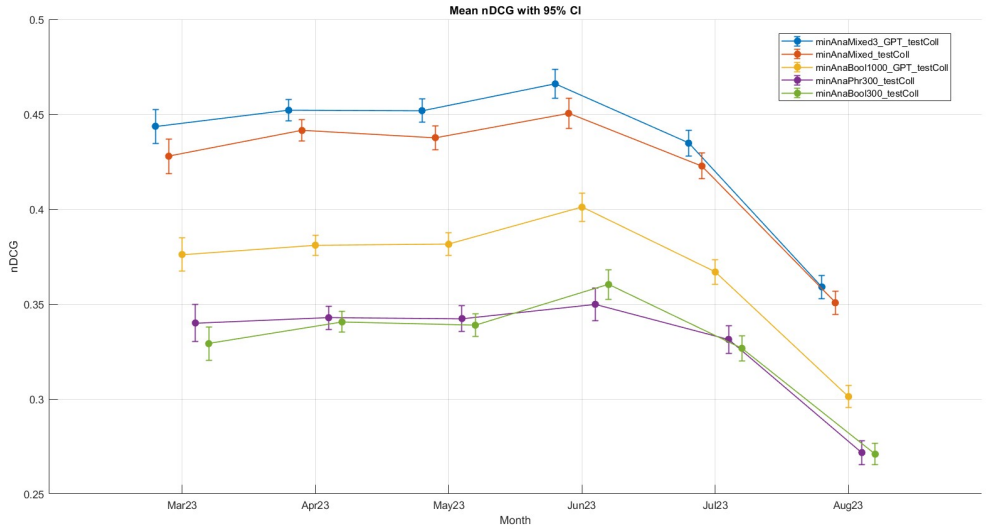


Figure 6: Mean nDCG over months with 95% C.I.

Table 4
Performance of the submitted runs (along with the baseline) on the LongEval test set

Run	MAP	nDCG
minAnaMixed3_GPT_testColl	0.3271	0.4346
minAnaMixed_testColl	0.3189	0.4218
minAnaPhr300_testColl	0.2667	0.3297
minAnaBool1000_GPT_testColl	0.2609	0.3680
minAnaBool300_testColl	0.2338	0.3278

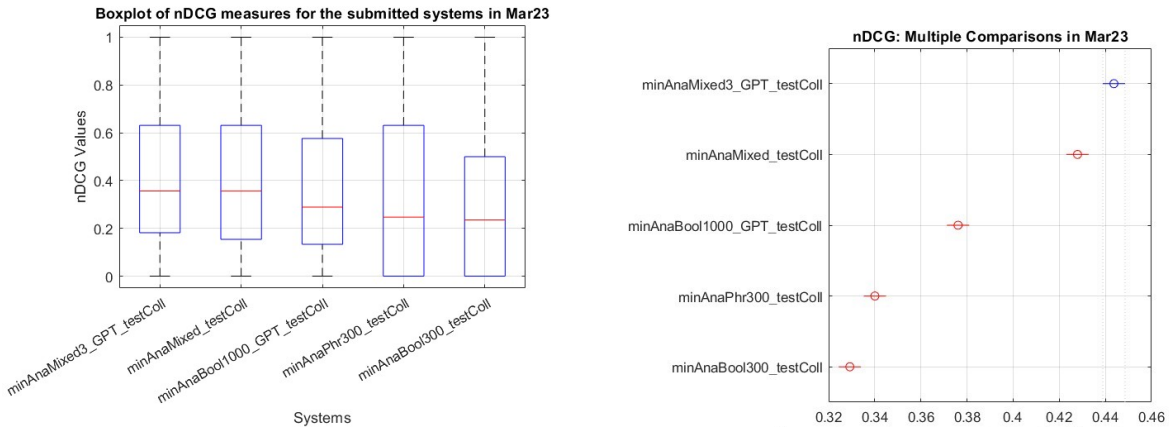
5.3.1. March 2023 Analysis

The analysis reveals significant differences in nDCG performance across systems, with systems using hybrid queries clearly outperforming others. Although by a small margin, the system using Phrase queries outperforms the baseline and the difference is statistically significant. In this initial month of the

test collection, all systems differ significantly from one another, as shown by the multiple comparisons (Figure 7b).

Table 5
Two-way ANOVA table for nDCG scores on test runs (Mar 23)

Source	SS	df	MS	F	Prob > F
Columns	34.4218	4	8.6055	247.4546	5.3437e-208
Rows	2.4983e+03	5166	0.4836	13.9061	0
Error	718.6089	20664	0.0348	–	–
Total	3.2513e+03	25834	–	–	–



(a) Boxplot of nDCG scores

(b) Multiple comparisons of nDCG scores

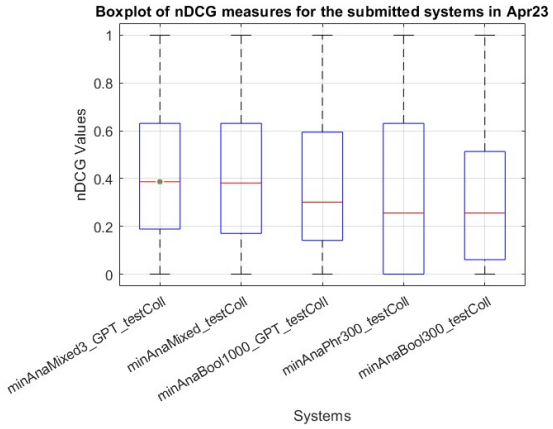
Figure 7: nDCG analysis on test runs for Mar 23.

5.3.2. April 2023 Analysis

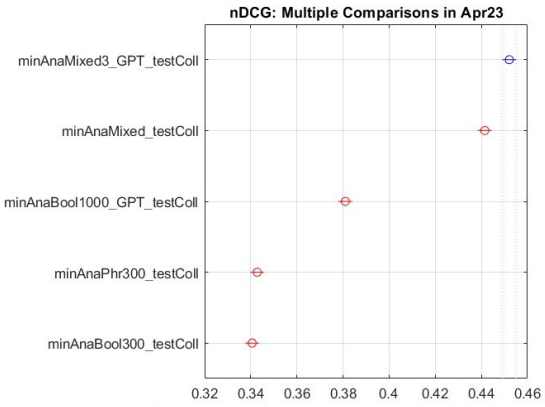
The second month of the test collection shows results largely consistent with the first, with the same relative ranking among systems. This confirms the trend already observed in the training set and reinforced by the March test results. The only notable difference compared to the previous month is that the system using Phrase queries is no longer statistically different from the baseline.

Table 6
Two-way ANOVA table for nDCG scores on test runs (Apr 23)

Source	SS	df	MS	F	Prob > F
Columns	95.0111	4	23.7528	698.9536	0
Rows	6.1365e+03	13102	0.4684	13.7822	0
Error	1.7810e+03	52408	0.0340	–	–
Total	8.0125e+03	65514	–	–	–



(a) Boxplot of nDCG scores



(b) Multiple comparisons of nDCG scores

Figure 8: nDCG analysis on test runs for Apr 23.

5.3.3. August 2023 Analysis

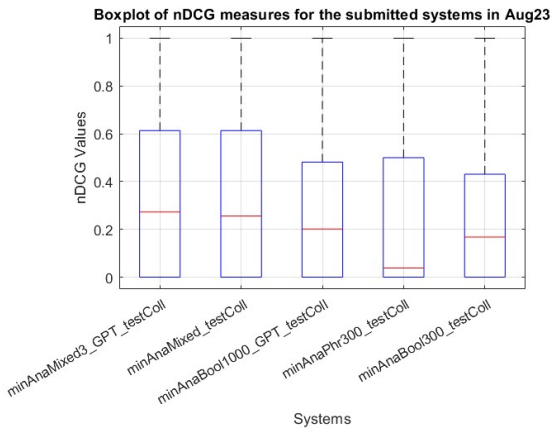
The final month of the test collection confirms the same relative ranking among systems and the significant performance gap between hybrid-query based systems and the others, highlighting the robustness of these approaches over time. As in April, the system based on Phrase queries is statistically indistinguishable from the baseline. It is also worth noting that all systems report lower nDCG scores in this month compared to the previous two analyzed, suggesting increased difficulty when the test set diverges more substantially from the training months on which systems parameters and configurations were tuned.

Overall, across the three snapshots of the test set (March, April, and August 2023), the results confirm the superior performance and robustness of hybrid-query systems combining Boolean and Phrase queries. Although all systems experienced some degradation in August, the snapshot most temporally distant from the training data, the relative ranking among systems remained consistent. Notably, the system using Phrase queries alone showed a marked drop in nDCG in August, with half of the topics scoring near zero, suggesting a particular sensitivity to temporal drift in topic-document relevance.

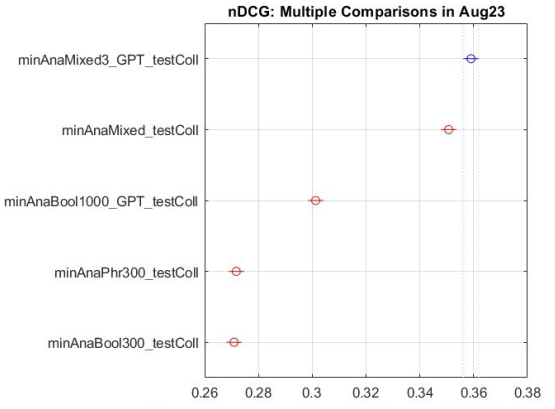
Another important observation concerns the system's performance with Phrase queries, as illustrated by the boxplot in Figure 9a. The position of the red line indicates that for half of the topics, the nDCG score is nearly zero, an outcome that markedly differs from the results observed in March and April.

Table 7
Two-way ANOVA table for nDCG scores on test runs (Aug 23)

Source	SS	df	MS	F	Prob > F
Columns	55.5128	4	13.8782	520.8715	0
Rows	4.9581e+03	11626	0.4265	16.0058	0
Error	1.2391e+03	46504	0.0266	–	–
Total	6.2526e+03	58134	–	–	–



(a) Boxplot of nDCG scores



(b) Multiple comparisons of nDCG scores

Figure 9: nDCG analysis on test runs for Aug 23.

6. Conclusions and Future Work

Throughout the project, we observed that even light query processing using GPT to correct errors in queries led to significant improvements when used on Boolean models. We found that combining Boolean queries with Phrase queries outperformed each method individually, confirming the added value of this hybrid approach. As we progressed through the various steps of development, we consistently saw improvements in retrieval performance.

However, query expansion, especially when using dictionaries, did not yield the expected benefits. We concluded that this approach was too generic, particularly for French, where available resources, such as WordNet for English, are not as rich. The limitations of French language tools ultimately led to suboptimal results in the query expansion process.

Overall, the project demonstrated that even small enhancements in query handling can lead to substantial gains in retrieval performance, but the choice of tools and approaches, especially for less-resourced languages like French, remains crucial for achieving optimal results.

Future work will focus on incorporating more advanced retrieval techniques that could not be explored in the current project due to hardware constraints and limitations in the available data. In particular, the use of LLMs for tasks such as query rewriting, expansion, and contextual understanding represents a promising direction. These models can enhance retrieval effectiveness by generating more semantically rich and intent-aware queries.

Another avenue we intend to explore is the application of supervised Learning to Rank methods such as LambdaMART, which have shown strong performance in retrieval competitions and industry applications. However, the effectiveness of such models strongly depends on the availability of reliable and fine-grained relevance judgments.

Overall, advancing toward these techniques will require both a more robust computational infrastructure and improvements in data quality. Nonetheless, their integration could significantly push the boundaries of performance within the LongEval framework and similar longitudinal retrieval settings.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4 via ChatGPT for two main purposes: (i) to assist in rewriting user queries in natural language during system development and evaluation; and (ii) to improve the grammatical accuracy and fluency of some textual sections of this manuscript. All content generated by the AI tool was thoroughly reviewed, edited, and validated by the authors, who accept full responsibility for the final submitted version, in compliance with the CEUR-WS policy on the use of generative AI tools. (https://ceur-ws.org/GenAI/Policy.html).

References

[1] CLEF LongEval, CLEF LongEval Tasks Overview, https://clef-longeval.github.io/tasks/, 2025. Accessed: 2025-04-19.

[2] 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.

[3] L. Bellin, A. A. Carè, M. Martini, M. T. Pepaj, M. Salvalaio, A. Segala, M. Tognon, N. Ferro, SEUPD@CLEF: Team GWCA on Longitudinal Evaluation of IR Systems by Using Query Expansion and Learning To Rank, in: Conference and Labs of the Evaluation Forum https://ceur-ws.org/Vol-3497/paper-186.pdf, 2023.

[4] R. Alkhalifa, H. Borkakoty, R. Deveaud, A. El-Ebshihy, L. Espinosa-Anke, T. Fink, P. Galuščáková, G. Gonzalez-Saez, L. Goeuriot, D. Iommi, M. Liakata, H. T. Madabushi, P. Medina-Alias, P. Mulhem, F. Piroi, M. Popel, A. Zubiaga, Extended Overview of the CLEF 2024 LongEval Lab on Longitudinal Evaluation of Model Performance , in: Conference and Labs of the Evaluation Forum https://ceur-ws.org/Vol-3740/paper-213.pdf, 2024.

[5] 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, Extended Overview of the CLEF-2023 LongEval Lab on Longitudinal Evaluation of Model Performance , in: Conference and Labs of the Evaluation Forum https://ceur-ws.org/Vol-3497/paper-184.pdf, 2023.

[6] Oracle Java Magazine, Java JSON Serialization with Jackson, https://blogs.oracle.com/javamagazine/post/java-json-serialization-jackson, 2022. Accessed: 2025-04-19.

[7] PeMISTAHL, Lingua: A Language Detection Library for Java and Kotlin, https://github.com/pemistahl/lingua, 2022. Accessed: 2025-04-19.

[8] Priceton University, About WordNet, https://wordnet.princeton.edu/, 2010. Accessed: 2025-04-26.

[9] B. Sagot, D. Fišer, Building a free french wordnet from multilingual resources, in: Ontolex 2008, Marrakech, Morocco, 2008.

[10] CLEF LongEval, CLEF LongEval Data Overview, https://clef-longeval.github.io/data/, 2025. Accessed: 2025-05-04.