Team RAND at LongEval: Composable Information Retrieval with Semantic and Language-Aware Components

Notebook for the LongEval Lab at CLEF 2025

Giorgia Amato[1,†], Nicolas Brentel[1,†], Alessio Demo[1,†], Steven Laghetto[1,†], Francesco Pivotto[1,†], Iegor Toporov[1,†] and Nicola Ferro[1]

[1] *University of Padua, Italy*

Abstract

This paper presents our work done by team RAND (University of Padua) on LongEval-Web Retrieval challenge, which investigates the robustness and stability of Web search engines in the context of evolving document collections.

Our team began by analyzing the dataset and applying standard IR techniques to establish baseline performance. We then iteratively refined our approach by focusing on methods that demonstrated improved effectiveness in handling temporal changes across snapshots.

Keywords

LongEval 2025, Information Retrieval, Query Expansion, Filters

1. Introduction

This report aims at providing a brief explanation of the Information Retrieval system built for Task 1. LongEval-Web Retrieval. The primary objectives are to assess how search engine performance is affected as the underlying Web data changes over time, and to determine when an information retrieval (IR) system requires updates in response to such changes. The task relies on a dataset includes a series of document and query snapshots that reflect the changing nature of the Web over time. The documents and queries were collected monthly, covering the period from June 2022 to February 2023, and are based on real-world user interactions from the Qwant search engine. The main idea of the RAND group was to design and implement a modular and extensible Information Retrieval (IR) system as part of the CLEF 2025 LongEval lab. The core of the system is a Java-based framework designed to handle the full IR pipeline (including parsing, indexing, searching, and evaluation) while supporting different document formats through a parser structure for processing both .trec and .json files (Section 2.2.2). To complement this, Python-based components were integrated for experimentation and benchmarking. Special attention was given to exploring the impact of preprocessing strategies (e.g., stopword filtering) and query variations on retrieval effectiveness, allowing the system to adapt to temporally evolving document collections. This dual-language approach provided both performance and flexibility in evaluating IR techniques over time.
The paper is organized as follows:
Section **??** introduces related works;
Section 2 describes our approach;
Section 3 explains our experimental setup;

CLEF 2025 Working Notes, 9 – 12 September 2025, Madrid, Spain
[†] These authors contributed equally.
giorgia.amato@studenti.unipd.it (G. Amato); nicolas.brentel@studenti.unipd.it (N. Brentel); alessio.demo@studenti.unipd.it (A. Demo); steven.laghetto@studenti.unipd.it (S. Laghetto); francesco.pivotto.1@studenti.unipd.it (F. Pivotto); iegor.toporov@studenti.unipd.it (I. Toporov); nicola.ferro@unipd.it (N. Ferro)
https://www.dei.unipd.it/~ferro/ (N. Ferro)
0000-0001-9219-6239 (N. Ferro)

Section 4 discusses our main findings;
Section 5 draws some conclusions and outlooks for future work.

2. Methodology

We developed this information retrieval system by applying the theoretical foundations covered in the course alongside the practical implementations introduced during the tutoring sessions. The development began with the creation of the `RANDDocument` class and the `RANDUniversalParser`, which together enable the conversion of document collections in different formats (`.trec` and `.json`) into a unified `RANDDocument` object. Subsequently, we implemented the `RANDAnalyzer` class, capable of performing tokenization, stopword removal, and stemming operations, configurable for multiple languages. We would like to highlight that the Vocabulary Statistics class and its implementation was inspired by the project created by a group from the same course two years ago, specifically Team GWCA[1] [1]. The next step involved designing the search component, which executes topic-based searches over the indexed collections using a query file and produces output in standard TREC format. The final stage of the pipeline is evaluation, carried out through the `Evaluator` class. This module leverages the `trec_eval` tool to compute standard information retrieval metrics such as the number of retrieved and relevant documents, DCG, and more. Finally, all components are integrated through the `Main` class, which orchestrates the execution pipeline with the support of the `FileCollector` utility—allowing for recursive file retrieval from nested directories.

2.1. Workflow and Foundations

2.1.1. Base System

The base information retrieval pipeline is structured into three primary stages: **parsing**, **indexing**, and **searching**. During the **parsing phase**, the system supports input in both JSON and TREC formats. Text content is preprocessed to remove unwanted elements such as HTML tags, hyperlinks, special characters, and other noise, ensuring a clean and consistent representation of the data. In the **indexing phase**, the cleaned documents are transformed into a searchable format using the Lucene indexing framework. This stage supports extensive customization of text processing operations, including tokenization, normalization, and similarity modeling, allowing flexibility in how documents are represented and ranked. The **searching phase** involves executing queries over the constructed index. The system loads a predefined set of queries and processes them using the same text analysis pipeline employed during indexing. Advanced techniques such as tokenization and synonym-based query expansion are supported to improve recall and relevance. Additionally, semantic enhancement tools can be integrated into the process to further refine the search results. Finally, the retrieved and ranked documents are exported in a standardized TREC format, ensuring compatibility with established evaluation tools.

2.1.2. Re-ranking

To enhance the precision of retrieval and promote more relevant documents to higher ranks, a re-ranking mechanism has been implemented within the `RANDSearcher` class. This step is optional and is triggered only when the top-scoring document from the initial Lucene-based search does not meet a configurable similarity threshold. The reranking phase functions as a post-retrieval adjustment, reevaluating the list of retrieved documents using semantic similarity techniques. After Lucene retrieves the initial list of documents (as `ScoreDoc[]` objects), the system computes semantic relevance between the query and the document bodies using one of two models: a Deep Learning (DL) model or a Large Language Model (LLM). The choice between these is defined in the system's `SearcherParams` configuration. The reranking process unfolds as follows:

[1]GWCA Team's repository: https://bitbucket.org/upd-dei-stud-prj/seupd2223-gwca/src/master/.

1. **Initial Retrieval:** A Lucene query is generated and executed using BM25 scoring. If synonym expansion is enabled, the query terms are augmented using data from the WOLF lexical database for French, and corresponding `SynonymQuery` terms are added.
2. **Reranking Trigger:** If the highest initial score is below a set threshold, re-ranking is activated. The threshold is used for time saving purposes, this addition is something counterintuitive for the standard re-ranking process, anyway the concept is the following:
   To optimize processing time in our search engine pipeline, we introduce a threshold-based mechanism in the re-ranking phase. Specifically, if a document receives a relevance score below 6 during the initial ranking stage, it is passed through the re-ranking process. Conversely, documents with a score equal to or above 6 are considered sufficiently relevant and are excluded from re-ranking. This strategy is based on the observation that high-scoring documents in the initial stage typically already align well with the user's query, making additional re-ranking unnecessary. By applying this threshold, we reduce computational overhead in terms of time, while maintaining high retrieval effectiveness.
   Each document in the top-k list is then semantically compared to the query.
3. **Semantic Scoring:**
   - In the **DL configuration**, a pre-trained sentence embedding model (e.g., Sentence Transformers) is used to compute vector representations and similarity scores.
   - In the **LLM configuration**, a large language model (such as GPT-like APIs) is employed to assess the query-document semantic match.
4. **Score Adjustment:** The semantic similarity score is scaled and added to the original Lucene score. This score fusion ensures that both lexical and semantic signals contribute to the final ranking. The following formula is applied:

   ```
   sd[i].score += (float)(5 * semanticScore[i]);
   ```

5. **Output Generation:** The re-ranked list is written to a TREC-compatible run file. Duplicates are avoided to maintain evaluation consistency.

This reranking module is particularly effective in addressing vocabulary mismatches and handling queries that require deeper semantic understanding. It is fully configurable and can be toggled between DL, LLM, or disabled modes via system parameters. Empirical results confirm that semantic reranking significantly improves early precision metrics (e.g., nDCG@10), without compromising the stability of the overall ranking.

Deep Learning Models For the deep learning reranking phase, we evaluated two pre-trained sentence embedding models: `multi-qa-MiniLM-L6-cos-v1` and `multi-qa-L6-cos-v1` [2], both optimized for multilingual semantic similarity tasks. While the `multi-qa-L6-cos-v1` model offers richer representations, its high computational cost made it impractical on large document sets. We therefore selected the lighter `multi-qa-MiniLM-L6-cos-v1`, which balances speed and semantic accuracy effectively. To keep runtimes reasonable, we limited reranking to the top 100 Lucene-retrieved documents. We observed a clear tradeoff: retrieving fewer than 100 documents yielded poor precision metrics (e.g., nDCG@10, MAP), whereas increasing the number of retrieved documents improved these metrics but caused runtime to increase exponentially due to the quadratic growth in pairwise similarity computations.

Large Language Model As an alternative, we explored reranking using the `unsloth/Llama-3.2-1B-Instruct` model [3], a Large Language Model designed for instruction-following tasks. In this setup, the system constructs a custom prompt that includes the query and candidate documents, and the LLM returns a relevance score for each pair. While the LLM provided advanced semantic reasoning, we ultimately excluded it from the final system due to its excessive runtime and high computational demands, especially when processing large document batches.

2.2. Implementation

This section provides an overview of the key classes developed to construct the information retrieval system. The final execution and orchestration of all components are managed by the `Main` class.

2.2.1. RANDDocument

The `RANDDocument` class serves as an abstraction layer for indexed and searchable documents. It is designed to handle both TREC and JSON input formats, providing a unified structure for the indexing pipeline. Upon ingestion, this class parses the raw data and converts it into a format compatible with the Lucene indexing engine. Specifically, it defines two main fields:

- **IdField**: A unique identifier for each document. This field is stored but not tokenized, ensuring fast and precise retrieval.
- **BodyField**: Contains the main textual content of the document. It is tokenized for full-text search and also explicitly stored to support potential re-ranking operations based on semantic similarity.

By supporting both TREC and JSON formats, the `RANDDocument` class allows the system to seamlessly process heterogeneous datasets while maintaining efficient indexing and retrieval.

2.2.2. RANDParser

Given the need to manage different input formats (i.e., JSON and TREC), we developed a flexible parser named `RANDUniversalParser.java`. This class dynamically selects the appropriate parser based on the input file's extension:

- If the file is in JSON format, it is handled by `RANDJsonParser.java`.
- If the file is in TREC format, it is delegated to `RANDTrecParser.java`.
- Files with unsupported extensions trigger an error message.

`RANDJsonParser.java` is based on the `WapoParser.java` presented during the tutoring sessions. It uses the Jackson library to process JSON content. The core method, `next()`, iterates through the collection and cleans each document by removing elements such as emojis, URLs, and HTML tags before returning it.
`RANDTrecParser.java`, derived from the `TipsterParser.java` example discussed in class, is a lightweight parser for TREC-formatted files. It identifies documents enclosed within `<DOC>` and `</DOC>` tags, extracts the document ID and content, and applies similar cleaning steps via the `cleanContent()` method.

2.2.3. RANDAnalyzer

The `RANDAnalyzer` class defines the text analysis pipeline used in our Information Retrieval system. It was designed to offer maximum flexibility during experimentation, making it a fully hybrid component capable of processing documents in both English and French. Thanks to the use of an external XML configuration file, all analysis parameters can be flexibly defined without modifying the codebase. This setup enables the selection of different tokenizers and filters depending on the chosen language configuration. The system supports three tokenizer types, configurable via the `TokenizerType` parameter:

- `WhitespaceTokenizer`
- `LetterTokenizer`
- `StandardTokenizer`

After tokenization, terms are passed through a series of filters:

- `LowerCaseFilter`: converts terms to lowercase,
- `LengthFilter`: removes terms based on length,
- `StopFilter`: eliminates stopwords.

Two main analysis pipelines were developed: one for English and one for French.

English configuration:

- `EnglishPossessiveFilter`: removes possessive suffixes.
- Followed by a selectable stemmer:
  - `EnglishMinimalStemFilter`
  - `PorterStemFilter`
  - `KStemFilter`
  - `SnowballFilter`

French configuration:

- `FrenchLightStemFilter`
- `ICUFoldingFilter`
- `ElisionFilter` (initially included but later excluded due to negative performance impact)

After extensive testing, the French pipeline—excluding the `ElisionFilter`—demonstrated the best retrieval performance, especially given that most relevant documents in the dataset were in French. These findings are further illustrated in the results shown in Section 4.2.

2.2.4. VocabularyStatistics

To verify index integrity and inspect the terms it contains, we initially considered using Luke (Lucene Index Toolbox). However, due to its limited usability for our specific needs, we explored alternative tools. Through prior projects, we identified a custom Java utility called `VocabularyStatistics`, developed by a previous student group (GWCA[2] [1]), which proved well-suited to our requirements. This utility generates a text file listing all terms indexed by Lucene, along with their frequency statistics. The main output file, `vocabulary.txt`, contains all terms sorted in descending order by raw frequency. Additionally, the tool can automatically append the most frequent terms to the stoplist (removing any duplicates). However, our testing showed that the manually curated `oracleFrench.txt` stoplist was already highly optimized. As a result, we chose to retain the original file and disable the automatic expansion feature, which had shown to degrade performance.

2.2.5. RANDIndexer

The `RANDIndexer` class is responsible for constructing a Lucene index from a collection of parsed documents. Its primary functions include:

- Creating the Lucene index based on `RANDDocument` objects produced by the parser.
- Avoiding duplicate entries and filtering out invalid documents.
- Enabling configuration of analyzers and similarity functions (e.g., BM25) to tailor the indexing behavior.

The indexer is composed of the following core components:

[2]GWCA Team's repository: https://bitbucket.org/upd-dei-stud-prj/seupd2223-gwca/src/master/.

Constructor Parameters:

- An `Analyzer` for tokenizing and processing document content.
- A `Similarity` model for scoring (e.g., BM25).
- An integer value defining the amount of RAM allocated for document buffering before flushing to disk.
- The path specifying where the index is stored.
- A `Parser` instance responsible for reading and transforming the source documents.

`index()` Method:

- Validates input documents, skipping null entries or those without a valid ID.
- Converts each `RANDDocument` into a Lucene `Document` using the parser's `toLuceneDocument()` method.
- Adds the documents to the index and finalizes the process with a commit and close operation.
- Outputs a report detailing the number of indexed, skipped, and duplicated documents.

2.2.6. RANDSearcher

The `RANDSearcher` class implements the search functionality of our system. It leverages Lucene's indexing infrastructure to retrieve documents relevant to a given set of queries in TREC format. Queries are read from a tab-separated file where each line contains a query ID and its corresponding description. These are analyzed using a `RANDAnalyzer`, ensuring consistency with the indexing phase. A `BooleanQuery` is then constructed, targeting the `Body` field of the indexed documents. Each query term is converted into a Lucene `TermQuery`. If query expansion is enabled, the `WolfManager` class provides synonym terms using the WOLF (Wordnet Libre du Français) resource. These synonyms are incorporated into the search via `SynonymQuery` objects with weighted contributions. The search process proceeds as follows.
The search process includes:

1. **Initial Retrieval**: Lucene search using BM25 (with optional synonym expansion).
2. **Optional Re-ranking**: Semantic similarity via DL or LLM models.
3. **Score Fusion**: The system combines lexical and semantic scores, as previously described in the Re-ranking section
4. **Output**: Results saved in TREC format, ensuring no duplicates.

This entire search and re-ranking pipeline is controlled by an XML configuration (`SearcherParams`). It supports customization of analyzers, similarity models, synonym expansion, and re-ranking methods. The `getReRanker()` parameter can be set to `DL`, `LLM`, or `None`, enabling fine-grained control over the system's semantic capabilities.

3. Experimental Setup

3.1. Collection

For our project, we utilized the **LongEval 2025 Web Retrieval Train Collection**, curated by TU Wien in collaboration with Qwant.
This dataset is designed to support research in temporal information retrieval and the evaluation of long-term relevance persistence.
Dataset Composition:

- **Queries:** 9,000 user-issued queries focused on trending topics, extracted from Qwant's search logs between June 2022 and February 2023.

- **Documents:** Approximately 18 million web documents retrieved in response to those queries, including both clicked results and randomly sampled content from Qwant's index.
- **Relevance Judgments:** Graded relevance labels derived from a click model, reflecting user interaction and engagement patterns.

Data Characteristics:

- The dataset includes original French versions of both queries and documents, adding a multilingual dimension to retrieval tasks.
- Documents are stored in JSON format, with metadata such as document ID, URL, and timestamp, enabling structured parsing and temporal analysis.
- The temporal coverage of the data captures language evolution and topical drift, challenging systems to maintain performance over time.

This collection is used as the official training set for the LongEval 2025 Information Retrieval Lab at CLEF and provides a robust benchmark for evaluating retrieval systems in dynamic and realistic web search scenarios.

3.2. Evaluation Measures

We evaluate the performance of our information retrieval system using three key metrics: **Mean Average Precision (MAP)**, **Normalized Discounted Cumulative Gain (nDCG)**, and **Normalized Discounted Cumulative Gain at rank 10 (nDCG@10)**.

- **MAP (Mean Average Precision)**: MAP provides an overall measure of the retrieval ability of the system to return relevant documents.
  It calculates the average precision for each query and then averages these values across all queries. This metric is useful for assessing how well the system ranks relevant documents throughout the entire result list.
- **nDCG (Normalized Discounted Cumulative Gain)**: nDCG evaluates the ranking quality of the top documents by giving more weight to relevant documents at the top of the result list.
  It is particularly valuable for assessing systems where the relevance of documents in higher ranks is more significant.
- **nDCG@10 (Normalized Discounted Cumulative Gain at rank 10)**: This variation of nDCG emphasizes the relevance of the top 10 documents.
  It applies a logarithmic discount to the relevance of documents at lower ranks, giving more importance to those ranked near the top.

We used the official `trec_eval` tool to calculate both MAP and nDCG metrics based on our system outputs and the provided qrels file.

The results will be discussed in detail in Section4.2.

3.3. Git

To develop the project, our group made use extensively of git in order to collaborate and organize the files.
The full source code is available on our BitBucket Repository.[3]

[3]RAND bitbucket repository: https://bitbucket.org/upd-dei-stud-prj/seupd2425-rand/src/master/.

3.4. Hardware

The used hardware involved different systems to manage the heavy workload on the machines more time-efficiently.
These were the machines that were used:

Table 1
Hardware specification of the four machines used in the experiments

Machine 1	
CPU	11th Gen Intel Core i7-1165G7
RAM	8GB 3200 MHz
OS	Windows 11
GPU	Intel Iris Xe Graphics

Machine 2	
CPU	AMD Ryzen 5 3600
RAM	16GB 3600 MHz
OS	Windows 11
GPU	NVIDIA GTX 1650 Super

Machine 3	
CPU	12th Gen Intel Core i5-12400F
RAM	16GB 3600 MHz
OS	Windows 11
GPU	AMD 6650XT

Machine 4	
CPU	AMD Ryzen 3 5300U
RAM	8GB 3200 MHz
OS	Ubuntu
GPU	Integrated Radeon Graphics

Machine 5	
CPU	8th Gen Intel Core i5-8250U
RAM	16GB 2400 MHz
OS	Windows 10
GPU	Intel UHD Graphics 620

4. Results

4.1. Runs description

Here we describe the configurations of our system and their results, with a summary of tokenizer, filters, stoplist, stemmer used for the runs.

Run ID	Tokenizer	Filters	Stoplist	Stemmer	Description
`seupd2425-rand-nofilters`	Whitespace	LowerCase, Length	None	None	Baseline configuration without stoplist or stemmer; standard analysis applied to all text.
`seupd2425-rand-englishFilter`	Whitespace	LowerCase, Length, English Possessive	Azure (Igor Brigadir)	Porter	Focus on English documents: possessive suffix removal, English stoplist, and Porter stemming.
`seupd2425-rand-queryLength`	Whitespace	LowerCase, Length	Oracle French	FrenchLight	French configuration with relaxed match threshold ($\geq$1 term) to improve retrieval for short queries.
`seupd2425-rand-frenchFilter`	Whitespace	LowerCase, Length	Oracle French	FrenchLight	Same French setup but with stricter query-doc match threshold ($\geq$2 terms) to improve precision.
`seupd2425-rand-DL`	Whitespace	LowerCase, Length	Oracle French	FrenchLight	Adds a Deep Learning re-ranker to prioritize relevant documents in top ranking positions.
`seupd2425-rand-synonyms`	Whitespace	LowerCase, Length	Oracle French	FrenchLight	Synonym support enabled with weights 0.5, 0.8, and 1; balances recall and precision, no re-ranker.
`seupd2425-rand-elision+synonyms0.5`	Whitespace	LowerCase, Length, Elision	Oracle French	FrenchLight	Adds ElisionFilter for French contractions and uses synonym weight = 0.5 for optimal results.
`seupd2425-rand-ICU`	Whitespace	LowerCase, Length, ICUFolding	Oracle French	FrenchLight	ICU Folding removes accents and normalizes Unicode; achieves the highest nDCG score.

Table 2
Detailed summary of system configurations and approaches tested.

4.2. Results on the training set

The table below presents the final averaged evaluation metrics for each system configuration (run). We report three key metrics: **nDCG**, **nDCG@10**, and **MAP**. These metrics provide a comprehensive view of the retrieval performance across different strategies applied during the experiments, including language filtering, synonym expansion, and reranking.
The reported values represent the average scores obtained over nine monthly runs. These results help to highlight the relative strengths of each configuration and guide the selection of the most effective retrieval strategy.
As discussed previously, the ICU configuration (2) was revealed to have the best performance in terms of each of the three evaluation measures considered for the comparison.
The FrenchFilter (2) runs turned out to have the second-best nDCG, while the re-ranking mechanism adopted in the DL configuration (2) leads to higher nDCG@10 and MAP values, as expected.

Table 3
Final averaged evaluation metrics (nDCG, nDCG@10, MAP) for each run

Run Name	nDCG	nDCG@10	MAP
`seupd2425-rand-nofilters`	0.1423	0.0951	0.0804
`seupd2425-rand-frenchfilter`	0.1638	0.1071	0.0950
`seupd2425-rand-englishfilter`	0.1223	0.0803	0.0679
`seupd2425-rand-elision+synonyms0.5`	0.1482	0.1004	0.0851
`seupd2425-rand-DL`	0.1401	0.1086	0.0972
`seupd2425-rand-synonyms0.5`	0.1475	0.1001	0.0849
`seupd2425-rand-synonyms0.8`	0.1393	0.0934	0.0789
`seupd2425-rand-synonyms1`	0.1318	0.0965	0.0762
`seupd2425-rand-ICU`	**0.1707**	**0.1170**	**0.0995**

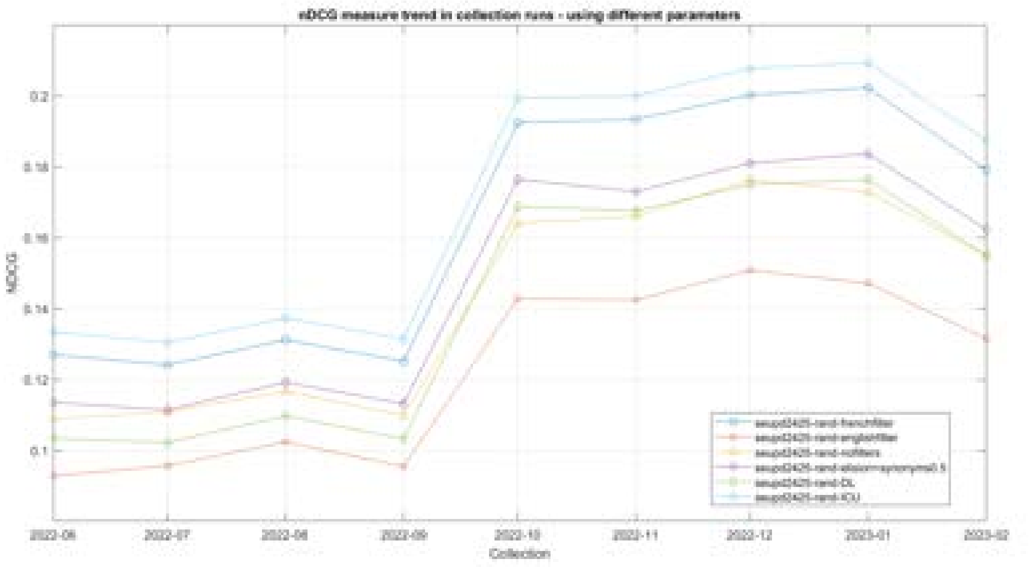

Figure 1: Comparison of nDCGs of different system configurations over the collection months.

The chart (Fig.1) shows the nDCG trend over time, evaluating the effectiveness of different parameter configurations in search runs; in particular, for re-ranking we retrieved a maximum of 100 documents per query, due to the execution time being too long to retrieve 1000 documents per query, this is also why the nDCG parameter is particularly low.

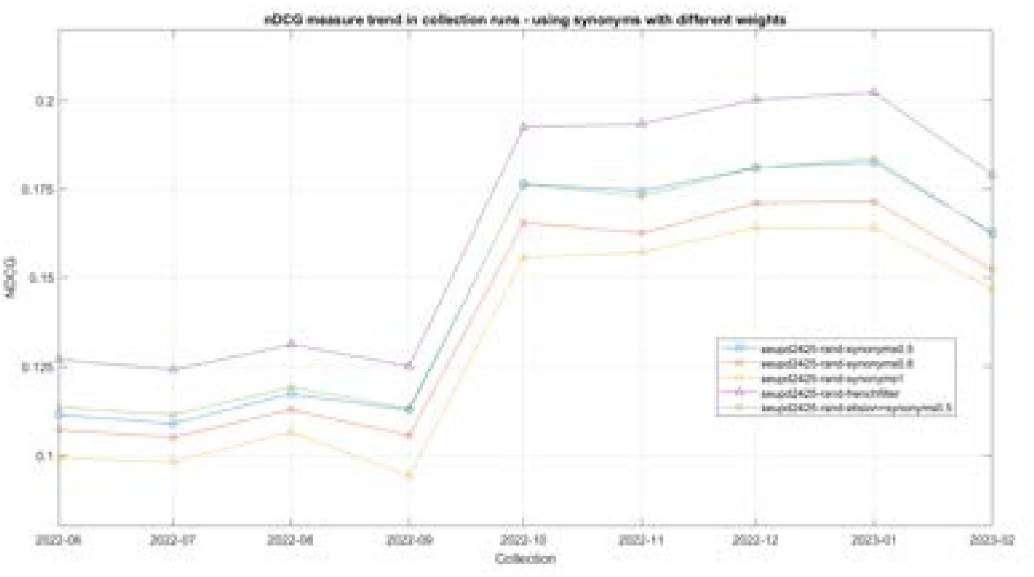

Figure 2: Comparison of system configurations using various synonym weights.

In particular, we can deduce the following trends from the graph:

- All configurations show a clear improvement starting around October 2022, probably due to the higher quality of the collections (e.g., better structured, less noisy), making it easier to return relevant documents.
- The most notable gains are seen with French-specific filters (2). Probably the reason is that the queries and documents are matched in the same language or tokenization style, reducing noise greatly and improving ranking quality.
- Without filters (2), irrelevant terms or tokens might be included in both indexing and querying stages, causing mismatches and drops in performance compared to runs with filters applied.
- reRanking (2) uses more sophisticated models (e.g., DL or LLM-based) to reorder the top results, improving the placement of highly relevant documents, but due to the fact that we set the parameter maxDocsRetrieved to 100, we have a drop in metrics.
- synonyms+elision (2) offers a moderate boost over noFilter, indicating that query expansion helps but is not as impactful as filtering.

The second graph (Fig.2) compares the different configurations of our system using various synonym weights (2) for query expansion. Runs with weight = 0.5 achieve better results in terms of nDCG, while increasing the weight leads to a drop in performance.
The comparison also includes the runs with the frenchFilter (2) and the elisionFilter (2), the latter using query expansion with a synonym weight of 0.5. Across all months in the collection, query expansion results in a degradation of system performance when compared to the frenchFilter configuration. The introduction of the `ElisionFilter` does not cause significant changes in the nDCG values compared to the weight = 0.5 configuration, but performs slightly better on average in terms of nDCG, nDCG@10 and MAP.

4.3. Results on the Test Collection

This section outlines the results of the runs submitted to CLEF, presenting both raw performance scores and accompanying statistical evaluations to allow for a clearer comparison of system behavior and the impact of the implemented strategies. As in Section4.2, we show the graph that represents the evolution of the nDCG over the months of the test collection, calculated for our four submitted systems

(Fig.3). This general comparison confirms that the ICU configuration (2) is the best in terms of nDCG. In order to follow the initial LongEval task, which consists in the development of an IR system capable

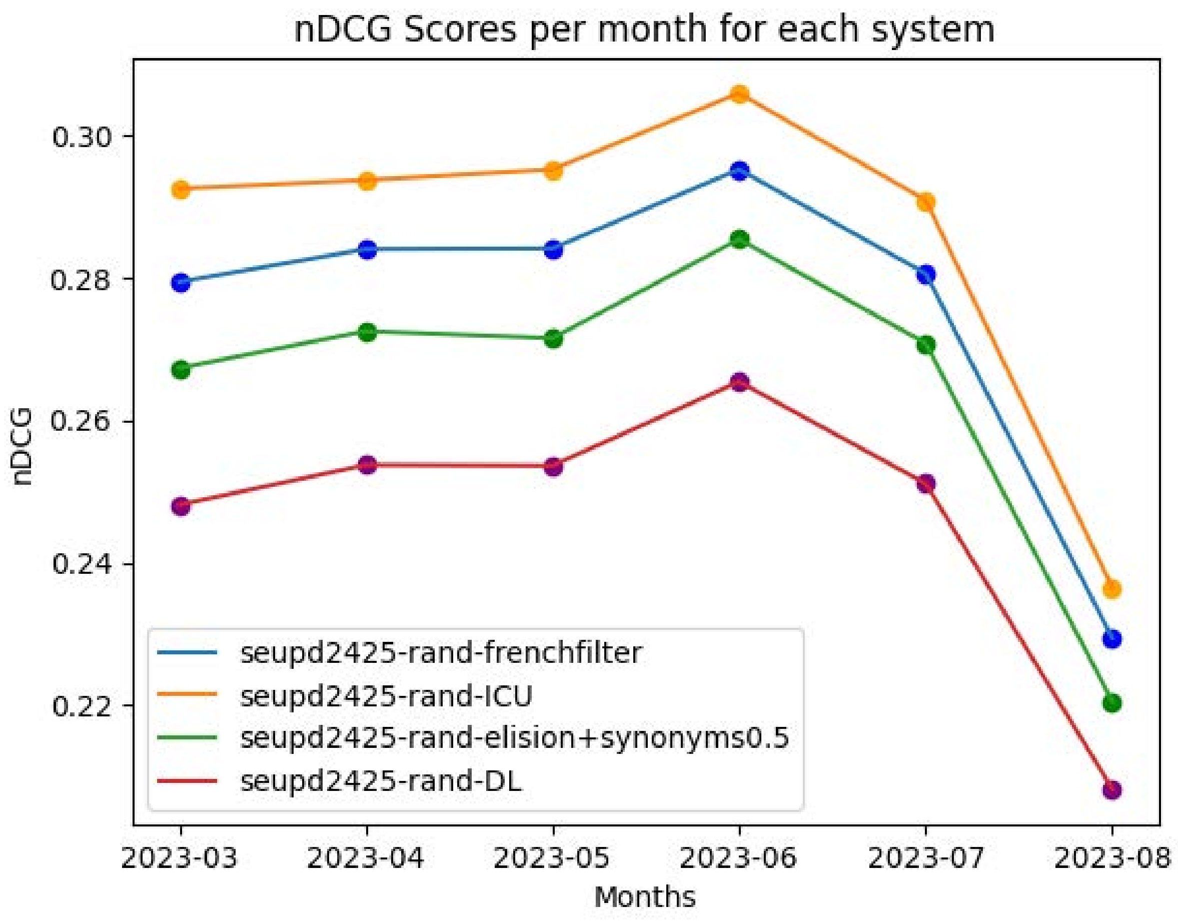

Figure 3: Comparison of nDCGs of the submitted system configurations over the collection months of the test set.

of managing data changes over time, the following analysis will focus on three different snapshots of the test collection: short-term (2023-03), middle-term (2023-06), and long-term (2023-08). For each of the three cases, we compare our submitted systems with boxplots, to visualize their mean and variance. After that, we compute the two-way ANalysis Of VAriance (2-ANOVA), considering as hypothesis the null hypothesis H_0: $\mu_x = \mu_y$ (where x and y represent two generic systems to be compared), and Multiple Comparisons with Tukey's HSD Test. All tests are performed with a significance level α = 0.05.

4.3.1. Short-term analysis

Just for the short-term analysis, we also show the results of the one-way ANOVA (Table 4). However, we will focus only on the two-way ANOVA for the remaining tests. The reason for our choice is that the two-way ANOVA considers both the topics and the systems' variabilities, while the one-way ANOVA only considers the system's variability.

Table 5 shows that both factors (systems and topics) are statistically significant (p < 0.001), with topics contributing substantially more to variance, indicating strong variability across queries. These considerations lead to rejecting the null hypothesis.

In Figure 5 we notice that the ICU system (in blue) is statistically different from the elision+synonyms0.5 and DL systems (in red), while it is not significantly different from the frenchfilter system (in gray).

Table 4
Results of the One-Way ANOVA test for 2023-03 month

Statistic	p_value	Reject null hypothesis H0
17.689	1.84	`Some systems have significant different means`

Table 5
Two-Way ANOVA results for nDCG on Short-term data.

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-value
Columns (systems)	4.9110	3	1.6370	25.6115	<0.0001
Rows (topics)	818.5685	4588	0.1784	2.7914	<0.0001
Error	879.7479	13764	0.0639		
Total	1703.2274	18355			

In figure 4 we can see that all systems show relatively low medians, around 0.15–0.2. and The IQRs are quite similar for all systems, indicating comparable variability.
Each system has positive outliers (near 1.0), meaning that a few instances achieved high effectiveness.

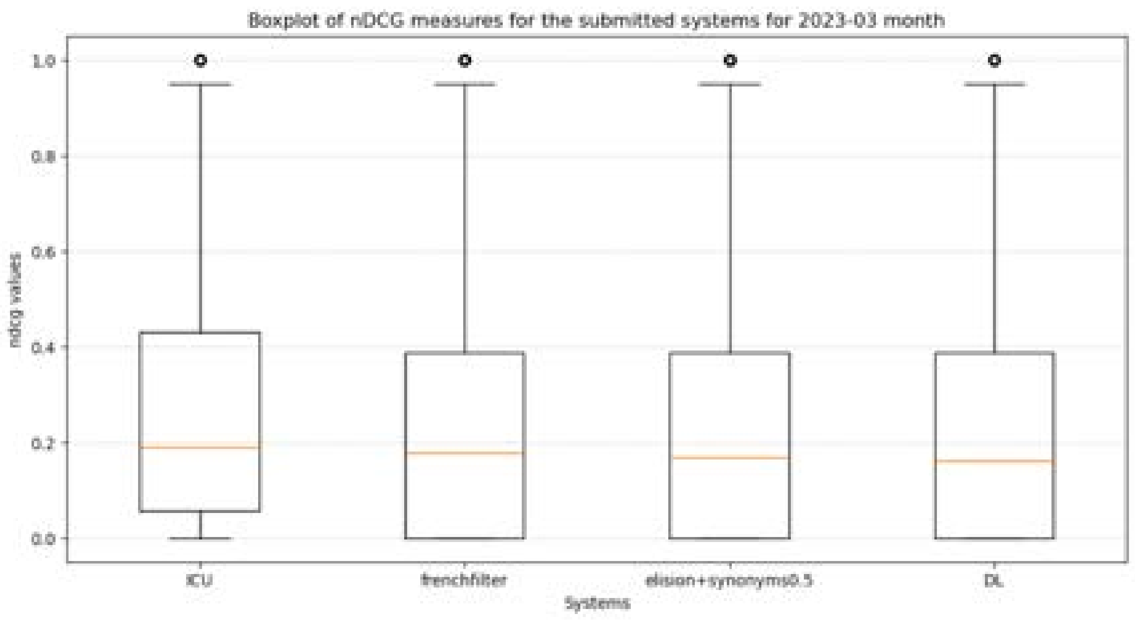

Figure 4: Short-term Boxplots

Table 6
Multiple Comparison of Means with Tukey HSD Test - Short-term (2023-03)

Systems	mean_diff	q_stat	p_value
`frenchfilter vs ICU`	0.0132	3.5438	0.0590
`frenchfilter vs elision+synonyms0.5`	0.0117	3.1438	0.1170
`frenchfilter vs DL`	0.0313	8.3869	**<0.0001**
`ICU vs elision+synonyms0.5`	0.0250	6.6876	**<0.0001**
`ICU vs DL`	0.0445	11.9307	**<0.0001**
`elision+synonyms0.5 vs DL`	0.0196	5.2431	**0.0012**

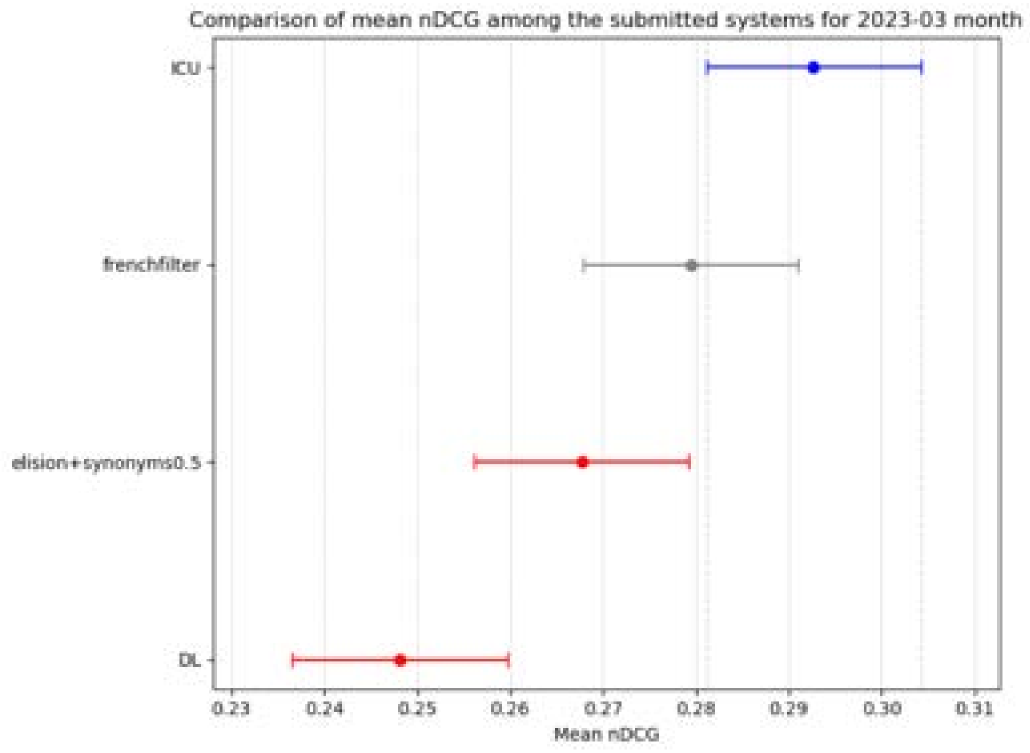

Figure 5: Short-term Tukey HSD Test

4.3.2. Middle-term analysis

In this second experimental setting, the results confirm that both the systems and the topics significantly affect performance, with p-values < 0.0001 for both factors (Fig.7). In particular, the F-value for the systems increased from 25.61 to 27.28, indicating a slightly stronger differentiation among the systems in this setting. In contrast, the F-value for the topics decreased from 2.79 to 2.26, though still statistically significant. The Sum of Squares for topics (1051.42) remains considerably higher than that of systems (5.83), showing that topic variability still accounts for the largest portion of the total variance. These findings support the conclusion that system performance differences are consistent and significant, even when accounting for the large variance introduced by the topic dimension.

Table 7
Two-Way ANOVA results for nDCG on Middle-term data.

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-value
Columns (systems)	5.8347	3	1.9449	27.2824	<0.0001
Rows (topics)	1051.4181	6513	0.1614	2.2645	<0.0001
Error	1392.8991	19539	0.0713	–	–
Total	2450.1520	26055	–	–	–

In Figure 6 median values and overall IQRs are fairly consistent across months, and the number of high-end outliers increased slightly, especially for elision+synonyms0.5, indicating more sporadic high-performance cases in comparison with the previous boxplot.

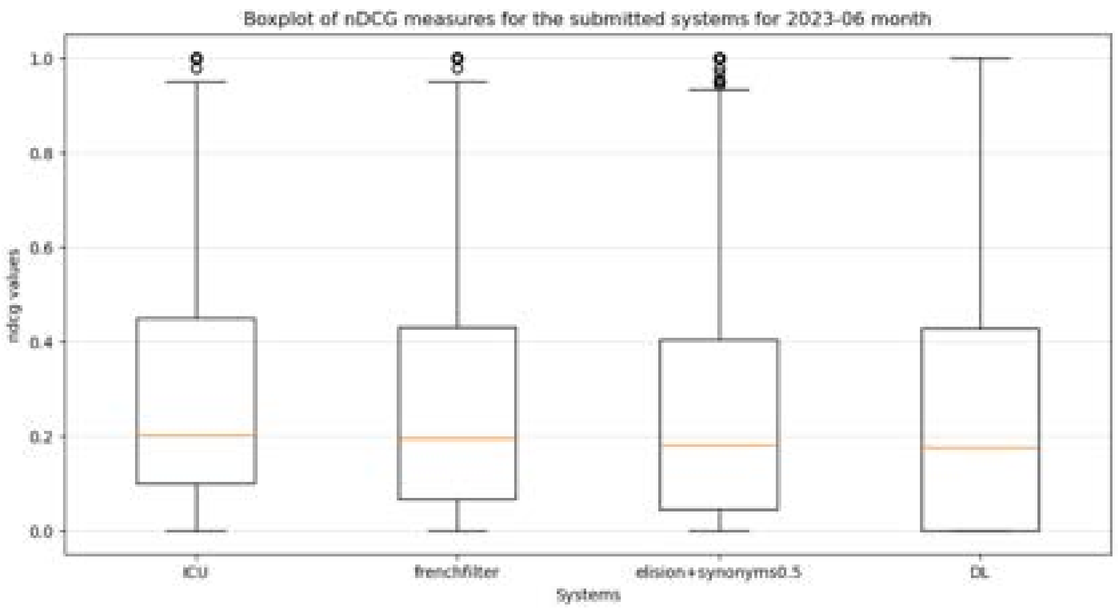

Figure 6: Middle-term Boxplots

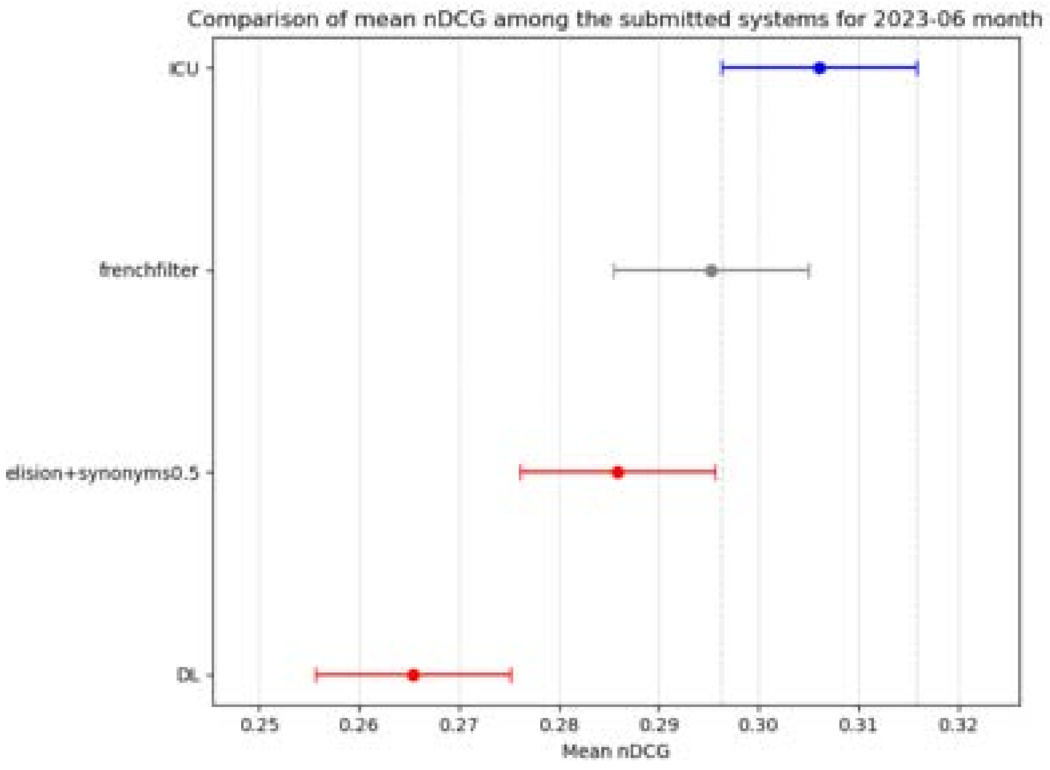

Figure 7: Middle-term Tukey HSD Test

Table 8
Multiple Comparison of Means with Tukey HSD Test - Middle-term (2023-06)

Systems	**mean_diff**	**q_stat**	**p_value**
`frenchfilter vs ICU`	0.0109	3.2877	0.0925
`frenchfilter vs elision+synonyms0.5`	0.0094	2.8504	0.1822
`frenchfilter vs DL`	0.0298	9.0178	**<0.0001**
`ICU vs elision+synonyms0.5`	0.0203	6.1381	**<0.0001**
`ICU vs DL`	0.0407	12.3054	**<0.0001**
`elision+synonyms0.5 vs DL`	0.0204	6.1674	**<0.0001**

4.3.3. Long-term analysis

The table 9 confirms a statistically significant difference among the systems and topics (p-values<0.0001). The Sum of Squares for topics is very high again (1456.32), which is consistent with the other tables (e.g., 1051.42 in the middle-term analysis). However, because the number of topics (df = 10554) is larger, the Mean Square is the smallest (0.1380) among all the different snaphots. The F-statistic for topics (2.1174) is slightly lower than the other monthly snapshots (e.g., 2.7914 in short-term snapshot), suggesting a significant effect of topic variability.

Table 9
Two-Way ANOVA results for nDCG on Long-term data.

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-value
Columns (systems)	4.7298	3	1.5766	24.1919	<0.0001
Rows (topics)	1456.3233	10554	0.1380	2.1174	<0.0001
Error	2063.4058	31662	0.0652		
Total	3524.4588	42219			

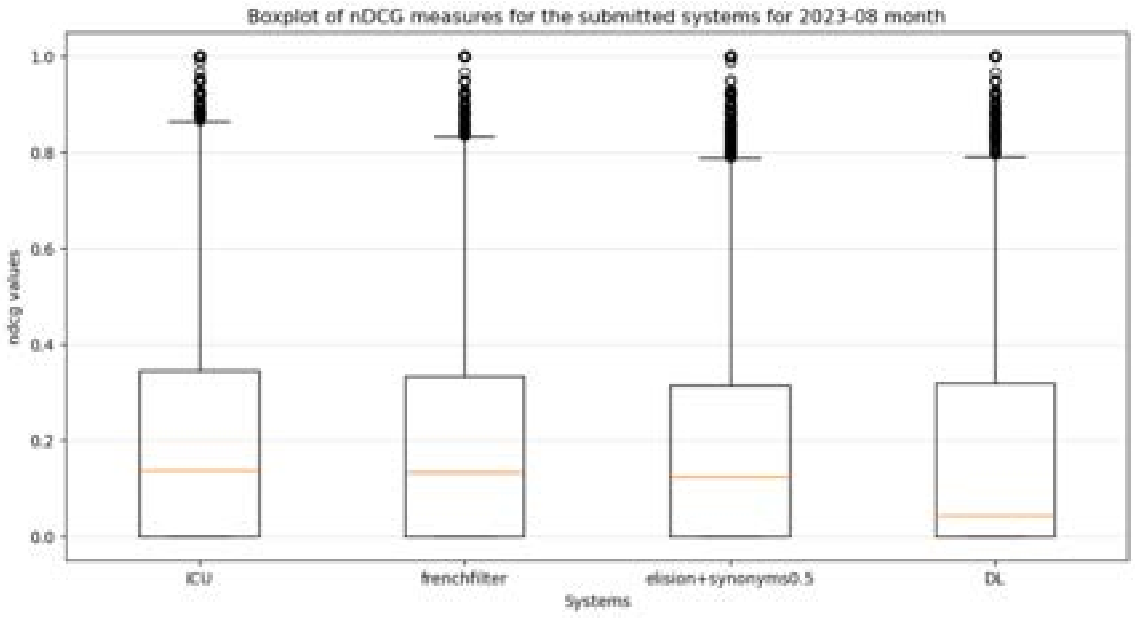

Figure 8: Long-term Boxplots

Finally in boxplot 8 we can also see that median values are lower than previous months, especially for DL, which shows a very low central tendency, close to 0.
All four systems have a large number of outliers, particularly concentrated near the top (nDCG near the '1' value). This suggests rare but highly successful retrievals.
The IQR is small, especially for DL, meaning most of its performance lies in a low range with little variability — except for many strong outliers.

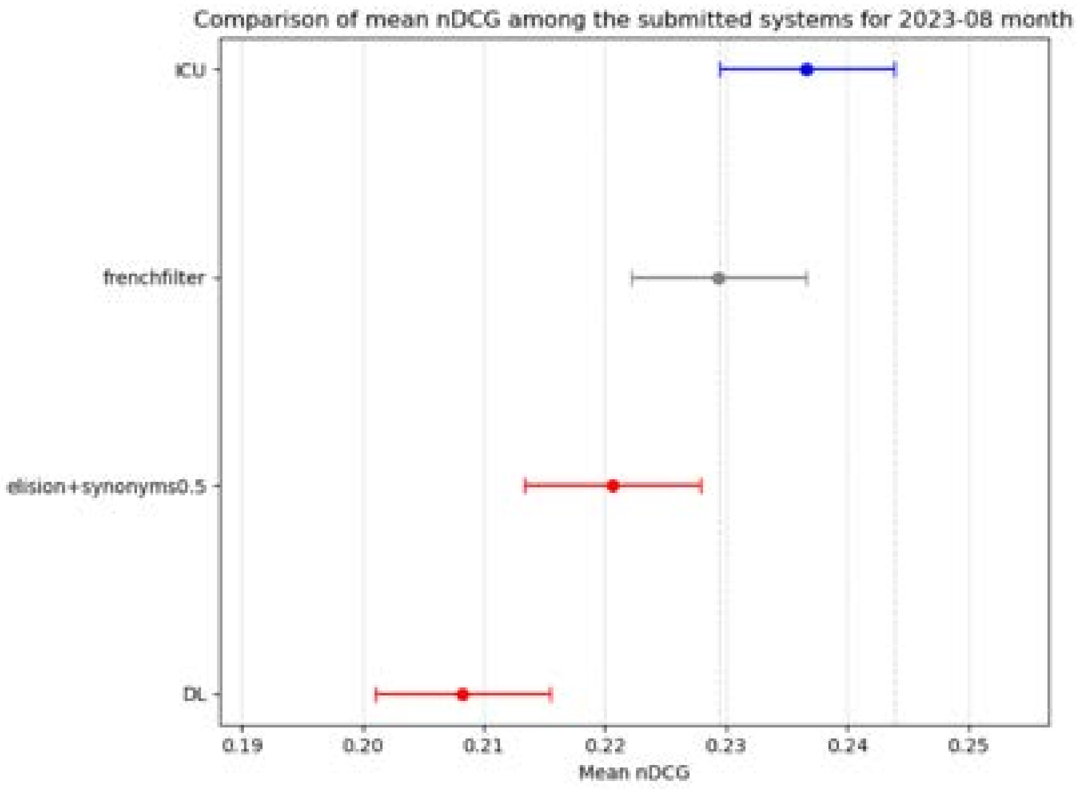

Figure 9: Long-term Tukey HSD Test

Table 10
Multiple Comparison of Means with Tukey HSD Test - Long-term (2023-08)

Systems	mean_diff	q_stat	p_value
`frenchfilter vs ICU`	0.0072	2.9125	0.1666
`frenchfilter vs elision+synonyms0.5`	0.0088	3.5330	0.0602
`frenchfilter vs DL`	0.0212	8.5126	**<0.0001**
`ICU vs elision+synonyms0.5`	0.0160	6.4454	**<0.0001**
`ICU vs DL`	0.0284	11.4251	**<0.0001**
`elision+synonyms0.5 vs DL`	0.0124	4.9797	**0.0024**

5. Conclusions and Future Work

This work presented a modular information retrieval system built on top of the Lucene framework, enhanced with custom query formulation strategies and optional query expansion through Wolf, using optionally deep learning or large language models for the re-ranking phase.
The system demonstrated flexibility in handling TREC and JSON topics and produced standardized run files suitable for evaluation.
Initial experimentation were based on few filters configuration, then French-dedicated stoplist and stemmer were introduced to improve our system performance.
The implementation of a Deep Learning model for reranking seemed to be promising in terms of nDCG@10, but its high computational cost forced us to retrieve a maximum of 100 documents per query, and this led to a drop of nDCG .
Future developments will focus on integrating deep learning models and large language models (LLMs) for more effective and efficient query rewriting and document re-ranking.
These enhancements aim to boost semantic understanding and retrieval precision.
Further work will also explore improved query expansion with synonyms, incorporating relevance feedback mechanisms, a better management of short queries, and refining scoring models to better capture user intent.

Declaration on Generative AI

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

References

[1] L. Bellin, A. A. Carè, M. Martini, M. T. Pepaj, M. Salvalaio, A. Segala, M. Tognon, N. Ferro, SE-UPD@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.

[2] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971 (2023).