SEUPD@CLEF: Team IRIS on Temporal Evolution of Query Expansion and Rank Fusion Techniques Applied to Cross-Encoder Re-Rankers.

Notebook for the LongEval Lab at CLEF 2024

Filippo Galli[1], Manuel Rigobello[1], Michela Schibuola[1], Riccardo Zuech[1,*] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This paper presents how the IRIS group of the Search Engines 2023/2024 course, held at the University of Padua, developed its Information Retrieval system for the Task 1 - LongEval-Retrieval of the LongEval CLEF 2024 Lab, aimed at developing an Information Retrieval (IR) system resilient to temporal evolution of Web documents. To address this challenge, our approach focused on traditional techniques combined with innovative methods. In particular, we focus on cross-encoder re-rankers and analyse *Generate, Filter & Fuse (GFF)* as a noise reduction technique to improve performance and resilience of *Query Expansion (QE)* techniques and re-rankers.

Keywords

CLEF2024, Search Engines, Information Retrieval, GFF, Q2K, Reranking, Cross-Encoder

1. Introduction

Search engines are software systems designed to help users find the best answers to their information needs, i.e. retrieving documents (web pages) that match as close as possible the information specified in a textual web search query. Search engines have become an essential part of the internet experience, and continue to evolve with it.

Research showed that the evolution over time of web documents may cause performance deterioration in IR systems as time passes. From this observation, the goal of our work is to develop systems that are resilient to performance losses over time.

Our systems are designed to retrieve documents from a corpus containing English and French files. We use ad-hoc techniques such as parsing, tokenization, stopwords removal and stemming to build a strong base foundation for our systems. Then, we explore novel approaches to query expansion by leveraging *Large Language Models (LLMs)* in keyword generation and re-ranking with sentence transformers, like in our implementation of the GFF technique.

Our paper is organized as follows: Section 2 introduces related work and our starting point for the development of our systems. Section 3 describes in detail the whole developed system and its capabilities. Section 4 shows our experimental setup. Section 5 discusses the experimental process we utilized to select our best systems, discussing useful insights and providing the results of our final systems on the train set. Section 6 introduces the results of our best systems in the test sets, finally providing deep analyses of their overall performances across time lags. Section 7 wraps-up the discussion by reporting statistical analysis to evaluate the effectiveness of our techniques and final systems. Section 8 draws our conclusions and outlooks for future work.

CLEF 2024: Conference and Labs of the Evaluation Forum, September 9–12, 2024, Grenoble, France

*Corresponding author.

filippo.galli@studenti.unipd.it (F. Galli); manuel.rigobello@studenti.unipd.it (M. Rigobello); michela.schibuola@studenti.unipd.it (M. Schibuola); riccardo.zuech@studenti.unipd.it (R. Zuech); nicola.ferro@unipd.it (N. Ferro)

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

0000-0001-9219-6239 (N. Ferro)

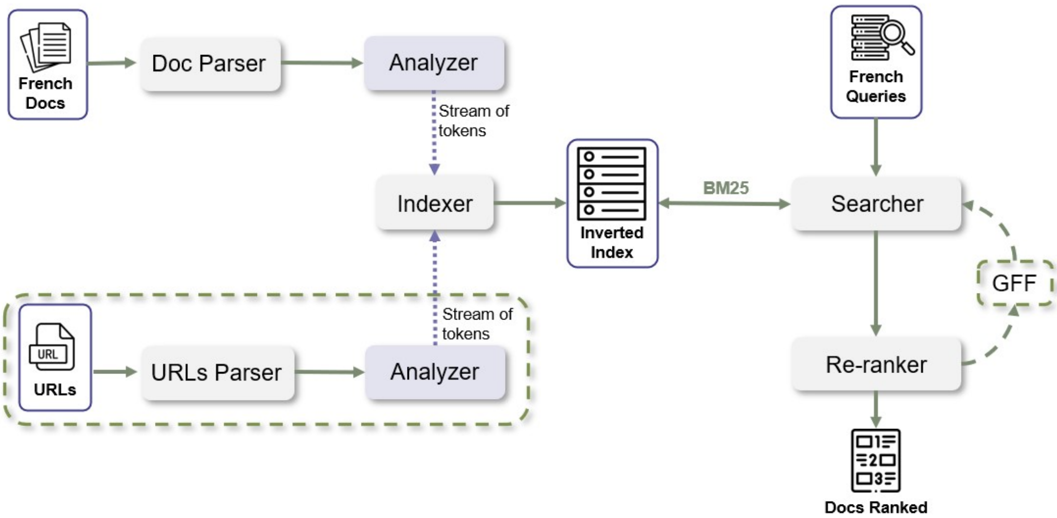


Figure 1: High-level view of the workflow of our systems. The dotted boxes indicate optional components that can be used or not by specifying them in the system parameters.

2. Related Work

The basic structure of our systems has been provided by Professor Ferro in the Search Engines course held at the University of Padua, it consists of an implementation of *Apache Lucene* components that we then adapted to our task and from which we built our systems on.

Additionally, we experimented with an adaptation of the GFF technique proposed by Li et al, 2023, v1 [1], from which we also took inspiration for the filtering mechanism used in our *Query to Keyword (Q2K)* implementation.

3. Methodology

In this section we describe the methodology adopted for the development of our system.

Our workflow began with creating a stable version of the system that could work with both English and French documents. After trying a few systems for both collections, we observed that those dealing with the French collection performed better, so we decided to focus on them. Since the English corpus consists of machine translations of the French one, the lower starting-point performance may be motivated by some translation errors & context loss.

Figure 1 represents an high-level view of the workflow of our systems, in the rest of this section we delve into the important features of the components.

3.1. Noise elimination: Parser components

Our systems implement a parser designed to handle content cleaning of documents, such as removing emojis, URLs, HTML, and code, based on various flags. Extensive details about our parser component can be found in Appendix A.

URLs Parser: Along with the above, a more robust URL parser, named `URLParser`, has been developed; it provides parsing for the docID to URLs mappings file, storing them in a hash map by segmenting the text of the URLs into words, removing prefixes and symbols specific to the URLs syntax. In particular, using `URLParser`, we removed all the syntactic constructs typical of URLs, such as extensions like .html, .php, .pdf, symbols like `#`, `-`, `=`, and split them in order to divide each element with a space.

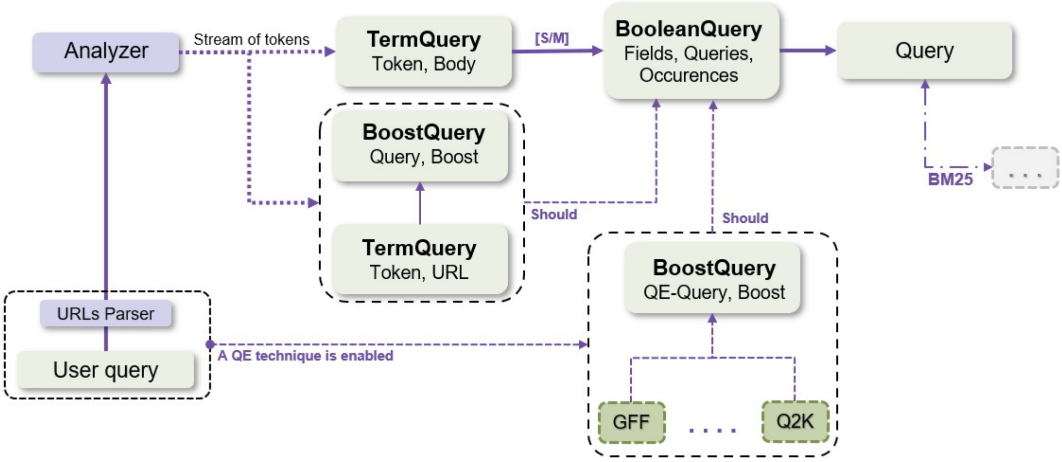


Figure 2: High-level description of the searcher component, responsible of building the queries and retrieving the associated relevant documents.

3.2. Text processing: Analyzer component

In our systems, text processing is handled by the analyzer component, whose role is to apply a selection of techniques such as tokenization, stemming, stopword removal, and more. Extensive details about our Analyzer component can be found in Appendix B.

3.3. Storing documents & URLs: Indexer component

In order to store the documents and allow fast retrieval at query time, we implemented an inverted index built by the `DirectoryIndexer` class, which is responsible for calling the `Parser` and `Analyzer`, retrieving documents processed by these components, and indexing them as instances of `ParsedDocument`. Additionally, if URLs support is enabled, the class also handles the mapping of documents to URLs and the storage of relative information in the inverted index.

BM25 The similarity function we selected for the first-stage retrieval is the *BM25*, as provided by the off-the-shelf Lucene's `BM25Similarity` [3], which allowed us to have a strong baseline and to move our focus to the novel techniques in the re-ranking phase.

3.4. Retrieving relevant documents: Searcher component

Figure 2 shows the basic workflow of our Searcher component.

The application of the URL parser (see Section 3.1) is motivated by the observation that many users search for partial URLs, e.g. *"emplois.inclusion.beta.gouv.fr"*.

After a first-stage retrieval step through *BM25*, the collection of retrieved documents may go through a re-ranking phase, to increase the quality of the order of the matches. The retrieve &/or re-rank phase can also be performed using more advanced techniques, such as our implementation of the GFF framework [1] and Q2K.

3.4.1. Re-ranker

With document re-ranking, we refer to the process of recomputing scores for a small subset of the retrieved documents, e.g. 1000, by a first stage retriever (in our case, *BM25*).

In particular, we focus our efforts on computing query-document similarities using multilingual bi-encoders and cross-encoders provided by the Python library Sentence Transformer [4].

We test two models:

- Cross-encoder: mmarco-mMiniLMv2-L12-H384-v1 [5], from now on referred to as MMARCO.

- Bi-encoder: paraphrase-multilingual-MiniLM-L12-v2 [6], from now on referred to as SBERT. For this model, we will produce two subsystems; see paragraph 3.4.1-"Query - Document similarity".

Using these models, we compute the new scores for the documents retrieved by BM25 with the following formula:

$$\text{score}_{\text{re-ranked}} = \text{score}_{\text{BM25}} + \alpha \cdot \text{sim}_{\text{Q-D}}$$

Where α is the weight parameter, specified by the `ScoreOperation` parameter of the searcher:

- `ADD`: take as α the value of the `ManualScoreWeight` parameter.
- `MAXMIN_ADD`: α is calculated on each query using the following formula:

$$\alpha = \max_{\text{doc} \in S_{\text{BM25}}} \text{doc.score} - \min_{\text{doc} \in S_{\text{BM25}}} \text{doc.score}$$

  where S_{BM25} is the set of documents retrieved by the BM25 first-stage retriever for each query.

Query - Document similarity $\text{sim}_{\text{Q-D}}$: Sentence transformers models, like the two we implemented, have a limited capacity in the number of tokens that they can receive in input. Taking as an example the MMARCO cross-encoder with parameter `max_length` $= 128$, if it receives in input a pair of strings whose total token count is greater than 128, it truncates both streams of tokens down to a maximum total sum of 128.

Taking into account this behaviour, we implemented a document-splitting solution by using a French model pre-trained in Apache OpenNLP Sentence Detector [7] to split each document at re-ranking time into a list of sentences.

Given the document as a list of sentences, we compute the query-document similarity by the following:

- SBERT, as a bi-encoder, produces vector embeddings of an input string, allowing us to build two subsystems:
  We first get the vector embedding of the query, then:
  - SBERTSUM: we get the document vector embedding as the sum of the sentence embeddings divided by the number of sentences; then we score the query-document similarity by cosine similarity between the query and document embedding.
  - SBERTAVERAGE: we get the list of sentence embeddings in order to compute a list of query-sentence similarities by cosine similarity, and, finally, we score the query-document similarity by taking the average of the query-sentence similarities.
- MMARCO, as a cross encoder, takes in input two lists of strings and produces a matrix of similarities between pairs from the two lists: we compute the query-document similarity by first producing a list of query-sentence similarities through the model (first input list contains only the query, while the second contains all the sentences), and then aggregate the results into the final query-document similarity score by taking the average of the query-sentence similarities.

An useful intuition on the different methods is given in Figure 3.

Summarizer Another technique we implemented is summarization of long documents for re-ranking by using the *Large Language Model (LLM)* mistralai/Mistral-7B-Instruct-v0.2 [8]. Anyway, this technique has proven completely unfeasible for our systems, since the overhead time required to summarize documents, even a small fraction of them, is excessive compared to the potential benefit the technique could have brought.

3.4.2. Query Expansion: Synonyms & Query2Keyword

Query expansion refers to the class of techniques used to rewrite queries by expanding their information content to better match relevant documents at search time.

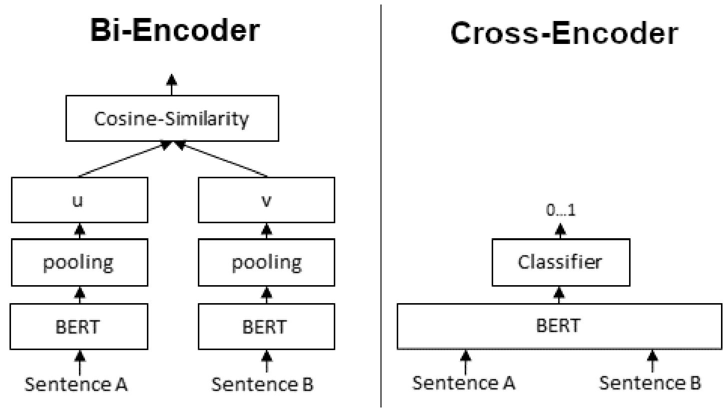


Figure 3: Graphical representation of the difference between bi-encoders and cross-encoders [6].

Synonyms: Given the user's query, we produce synonyms of the words composing the query and add both the original query and a weighted synonym `BoostQuery` to a Lucene's `SynonymQuery`, that will act as our original query to retrieve documents for. Analysing the results of the synonyms generation process, we observed that the produced synonyms are of low quality and dampen the overall performance of the systems, thus in our final systems this technique is not used.

Q2K: As a query expansion technique, we focus our effort on the Q2K technique using LLMs with few-shot prompting to generate a set of keywords from a query that might be relevant to the user's information needs.

Our *query to keyword (q2k)* implementation can be divided into the following steps:

1. Generate: for each query, we use the LLM to generate approximately 10 keywords related to that query. The process of keyword generation is repeated 3 times for each query, since LLM-based keyword generation methods involve stochasticity, and we want to filter out noise in the next step. In the end, we obtain up to approximately 30 keywords per query.
2. Filter: Given the query and its list of keywords, we first produce the tokenization of the query through our `Analyzer`, then we also tokenize each keyword, and if its tokens are different from those forming the query, we add the keyword to a frequency map. Finally, from the frequency map we take the top-k keywords by frequency (where k is specified by the parameter `QETopK`).
3. Boost: Given the top-k keywords by frequency, we produce a Lucene's `BooleanQuery` by adding at first the user's query and then each keyword as a Lucene's `BoostQuery` with weight specified by the parameter `keywordsWeight`.

As LLM, we used the model mistralai/Mistral-7B-Instruct-v0.2 [8] with a few-shot prompt with examples taken from MSMARCO passages. For the French system, we translated the examples into French to achieve better quality keywords. The French and English prompts used are reported in Appendix C.

Our system allows QE by Q2K to be performed both on-the-fly at search time or by using a pre-processed set of expanded queries. To generate keywords on-the-fly, we provide a Python server `mistralai_server.py`. To generate the pre-processed queries file, we provide the `MistralKWGF.py` Python script that adds to each query, given in a TREC-format TSV file, the list of keywords generated by the LLM.

Our q2k system is provided with support for both English and French queries, but, as already highlighted, we advise to focus on French, as it significantly outperforms its English variant.

Lastly, in case a system with URLs support is run with Q2K, the top-k keywords are also added as weighted *URLqueries*.

3.4.3. GFF: Generate, Filter & Fuse

Our system implements a modified version of the GFF [1] framework, adapted to our use case with some specific tweaking.

The strength of this technique lies in two aspects:

- The ability to generate high-quality keywords, leveraging proven effective techniques such as Q2K by LLMs and keyword filtering.
- The ability to fully utilize the increased performance provided by cross-encoder re-rankers (compared to bi-encoders), thanks to the high quality of keywords and the fusion of corresponding expanded queries results.

Generate: The first step is to generate keywords for a user's query using LLMs; this is done by using the same Q2K implementation described in the *Generation* step of Section 3.4.2.

Filter: The second step consists of the following:

1. Production of the top-k keywords: as already described in the *Filtering* step of Section 3.4.2, we produce a list of top-k keywords, where k is specified by the `QETopK` parameter.
2. Creation of expanded queries: we take the list of top-k keywords and produce a list of k Lucene's `BooleanQuery`s, where each expanded query is built from adding the original query and a `BoostQuery`, containing the keyword with weight `keywordWeight`, to the relative `BooleanQuery`.

Fuse: Finally, the fuse step consists of the following:

1. Search: We perform *BM*25 on the original query and store the retrieved documents in a list ordered by score L_{og}, then we search each expanded query, and store the result in a ordered list L_k, where k refers to the k-th expanded query.
2. Re-rank: With the objective of re-ranking the maximum number of documents, and observing that there is often a significant overlap in the results from the original and expanded queries, we maintain a hashmap to record the similarities between the original query and the documents that appear in L_{og} and each L_k. This hashmap has the role of speeding up computation by reusing similarities that have already been computed; this implies a noteworthy alteration from the original GFF framework [1], since we score the document similarities in each L_k to the original query instead of the respective expanded query.
   We first populate the hash map by re-ranking the first `maxDocumentsToRerank` documents in L_{og} and store the similarities returned by the chosen re-ranker. Then, given an internal document counter starting at `GFFLeftoverDocumentsLimit`, for each L_k in frequency order, we re-rank the documents that have already been scored for similarity, then compute and store similarities for first-time appearing documents up to what is left in the document counter, finally re-ranking them; in case the counter goes under `GFFMinimumDocumentsToRerank`, for each subsequent L_k, scores for new documents are computed up to `GFFMinimumDocumentsToRerank` documents. Finally, after updating the scores for each L_k, the list is ordered by decreasing document score.
   To get a better grasp of the hashmap population process, let us take as an example:
   - `maxDocumentsToRerank` $= 1000$,
   - `GFFLeftoverDocumentsLimit` $= 400$,
   - `GFFMinimumDocumentsToRerank` $= 100$

   We re-rank L_{og} for the first 1000 documents, then for each L_k (in order of keyword frequency), we compute the query-document similarity for each document that has not already been scored and store its similarity score by up to the first 400 new documents; After 400 new documents scored among expanded queries' lists, we score exactly up to 100 not-already-seen documents for each of the remaining lists.

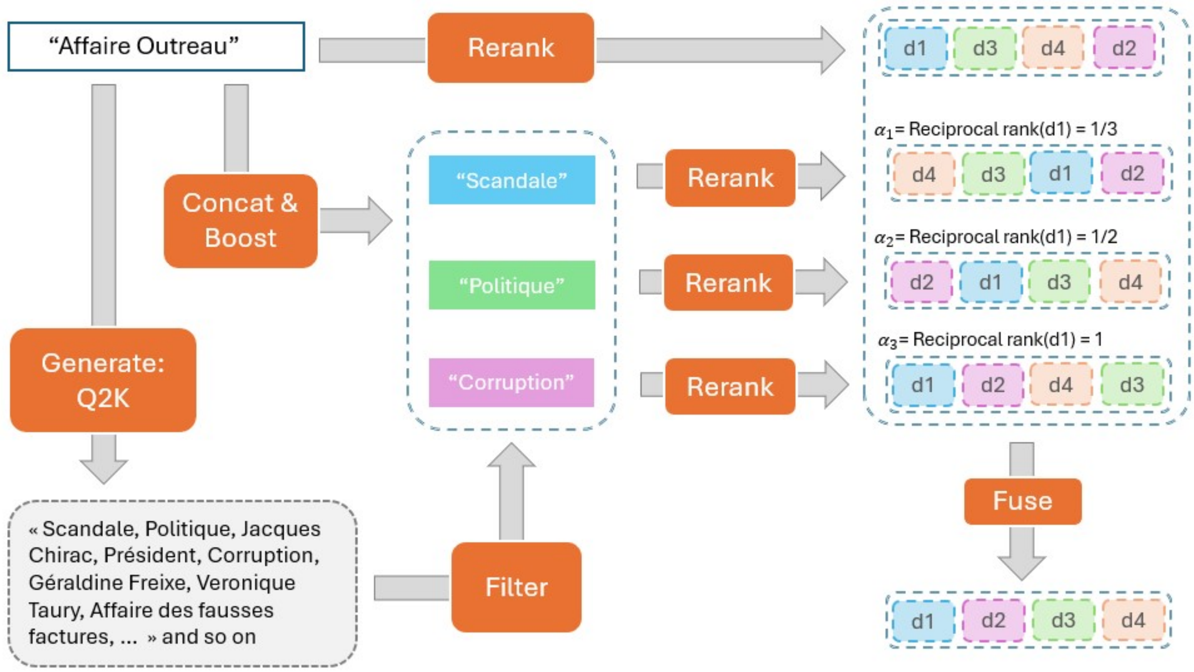


Figure 4: An intuitive representation of our implementation of the GFF[1] framework.

3. Weight: We associate with each L_k a weight α_k using the reciprocal rank [9] of the original top-1 document in this ranking list:

$$\alpha_k = \frac{1}{\text{Rank}(d^+, L_k)}$$

   where $\text{Rank}(d^+, L_k)$ is the rank of the top-1 document retrieved for the original query in the candidate list L_k associated with the expanded query k.

4. Fuse: We compute new scores for all the documents retrieved, using the following formula:

$$\text{score}_{\text{final}} = \text{score}_{\text{og}} + \sum_{k=1}^{\text{QETopK}} \alpha_k \text{score}_{\text{k}}$$

   Where score_{og} is the document score in L_{og} and score_{k} is the score of the document in L_k; the default value for both is 0 if they do not appear in the respective list.
   Finally, we take the top-`MaxDocumentsToRetrieve` in decreasing score order.

An high-level intuition on the basis of the above steps is given in Figure 4.

Lastly, in case a system with URLs support is run with GFF and the parameter `excludeURLsFromExpansion` is off, then the k keywords are also added as weighted *URLqueries* when retrieving L_k.

3.4.4. Available parameters

The list of parameters for our `Searcher` is available in the file *SearcherParams.xml* and, grouped by category, includes the following:

- General:
  - **ogQueryClause**: what Lucene's `BooleanQuery` clause to use for the original user query.
  - **useURLs**: whether to search for the user's query in the documents URL field or not; creates additional *URLqueries* added to the final `BooleanQuery`. It must be coupled with a URL-supporting index, refer to Section 3.3 & Section 3.1.
  - **urlsWeight**: weight to be assigned to the *URLqueries* as instances of Lucene's `BoostQuery`.

- Reranker:
  Refer to Section 3.4.1 for more details.
  - **reranker**: which re-ranker model to use, requires the corresponding `<model>-server.py` up and running. If the field is left empty, the re-ranking phase is skipped. Options are *MMARCO, SBERTAVERAGE, SBERTSUM*.
  - **maxDocumentsToRerank**: upper bound on the number of documents to be re-ranked after the first-stage search step.
  - **splitDocuments**: whether to split documents into sentences to provide as inputs to the re-ranker models.
  - **scoreOperation**: what formula to use for re-ranking documents. Possible values are *ASSIGN, ADD, MAXMIN_ADD* but, since *ASSIGN* provided very poor performances, it is omitted from this work.
  - **scoreManualWeight**: weight to be used in the *ADD* score operation.
  - **useSummarizer**: whether to invoke mistralai/Mistral-7B-Instruct-v0.2 [8] to summarize long documents (requires `mistral_server.py` up and running).
  - **tokenThreshold**: the minimum number of tokens in a document for the summarizer to be invoked.
- Synonyms:
  Refer to Section 3.4.2 for more details.
  - **useSynonyms**: whether to generate synonyms for the query.
  - **WordNetSynonymsPath**: path to the wordnet file for synonyms generation.
  - **WordNetSynonymsWeight**: weight to be assigned to the synonyms as a Lucene's `BoostQuery`.
- Q2K with LLMs:
  Refer to Section 3.4.2 for more details.
  - **queryExpansionTechnique**: what query expansion technique leveraging LLMs to use, the only possibility implemented is *Q2K*. The system does not use Q2K if the field is left empty.
  - **useProprocessedQueries**: whether to use a preprocessed TSV format file containing all expanded queries.
  - **preprocessedQueriesPath**: the path to the preprocessed expanded queries file.
  - **QETopK**:
    * Standard Q2K: the number of keywords to add the the original query.
    * GFF: the number of keywords to use in the framework; refer to Section 3.4.3 for more details.
  - **keywordsWeight**: weight to be assigned to the keywords as a Lucene's `BoostQuery`.
  - **rerankWithKeywords**: whether to concatenate the original query with its related keywords in the re-ranking phase. Since this reduced the systems performances, for the rest of this work we implicitly assume that it is set to `False`/0.
- GFF:
  Refer to Section 3.4.3 for more details.
  - **useGFF**: whether to use the adapted GFF framework implementation.
  - **GFFLeftoverDocumentsLimit**: upper bound counter for new documents to be re-ranked from the expanded queries first-stage search results.
  - **GFFMinimumDocumentsToRerank**: minimum number of new documents from the first-stage search results of expanded queries to always be re-ranked.
  - **excludeURLsFromExpansion**: whether to exclude *URLqueries* of the keywords in the expanded queries.

4. Experimental Setup

The goal of the project was to create one or more information retrieval systems that, for each given query, returned the most relevant documents related to that query. For that purpose, we needed:

- A repository that contained our IR systems, available at https://bitbucket.org/upd-dei-stud-prj/seupd2324-iris/src/master/.
- One or more machines to develop and run our systems, in particular the specifics of the machines we used are reported in Table 1.

Table 1
Machines used to develop and run our systems.

#	OS	CPU	RAM	GPU
1	Windows 11 Pro	AMD Ryzen 7 3750H	16GB 2400MHz	NVIDIA GeForce GTX 1660
2	Windows 11 Pro	AMD Ryzen 5 6600H	Crucial DDR5 16GB 4800MHz	NVIDIA rtx 3050 4GB
3	Windows 11 Pro	13th Gen Intel Core i9-13905H	32GB LPDDR5X-6400MHz	NVIDIA GeForce RTX 4060 8GB
4	Windows 11 Pro	AMD Ryzen 7 3700x	CORSAIR Vengeance 32GB 3600MHz	NVIDIA rtx 2060

4.1. Data description

LongEval released two sets of data containing queries and documents: a Training Collection and a Test Collection.

4.1.1. Train Collection

The train collection was given with the purpose of testing different techniques and choosing the systems with best results.
The detailed description is given by the LongEval website: "The collection consists of queries and documents provided by the Qwant search Engine (https://www.qwant.com). The queries, which were issued by the users of Qwant, are based on the selected trending topics. The documents in the collection were selected with respect to these queries using the Qwant click model. Apart from the documents selected using this model, the collection also contains randomly selected documents from the Qwant index. All the data were collected over January 2023. In total, the collection contains 599 train queries, with corresponding 9,785 relevance assessments coming from the Qwant click model. The set of documents consist of 2,049,729 downloaded, cleaned and filtered Web Pages. Apart from their original French versions, the collection also contains translations of the webpages and queries into English. The collection serves as the official training collection for the 2024 LongEval Information Retrieval Lab (https://clef-longeval.github.io/) organised at CLEF." [10]

To summarize the collection details:

- Collection 2023_01
  - Number of queries: 599;
  - Number of documents: 2049729.

Finally, CLEF also provided us also with the DocID to url mappings for the train collection.

4.1.2. Test Collection

The test collection was given with the purpose of running our systems through a larger set of queries and documents to produce the final results.
The detailed description is given by the LongEval website: "The collection consists of queries and documents provided by the Qwant search Engine (https://www.qwant.com). The queries, which were issued

by the users of Qwant, are based on the selected trending topics. The documents in the collection were selected with respect to these queries using the Qwant click model. Apart from the documents selected using this model, the collection also contains randomly selected documents from the Qwant index. All the data was collected over June 2023 and August 2023. In total, the collection contains 1,925 test queries. The set of documents consist of 4,321,642 downloaded, cleaned and filtered Web Pages. Translations of the webpages and queries into English will be added when available. The collection serves as the official test collection for the 2024 LongEval Information Retrieval Lab (https://clef-longeval.github.io/) organised at CLEF." [11]

To summarize the collections details:

- Collection 2023_06
  - Number of queries: 407;
  - Number of documents: 1790028.
- Collection 2023_08
  - Number of queries: 1518;
  - Number of documents: 2531614.

Finally, CLEF provided us also with the DocID to url mappings for the test collections.

4.2. Evaluation measures

To select the best results, the following evaluation measures have been taken into account: Precision at Document Cut-off (P@k), Mean Average Precision (MAP), Recall at Document Cut-off (R@k) & Normalized Discounted Cumulated Gain (NDCG), with a particular focus on the latter.

Our measures were computed through trec_eval [12], executed on the runs produced by our IR systems.

To aid comparisons between runs, we also created a Jupyter notebook implementing the Ranx [13] library, which allows to produce comparison tables for the above measures.

5. Experimental results: tuning alternatives on training data

In this section, we provide a summary of the experiments that lead us to choose our final runs to submit, as well as related observation and insights extracted from them.

The name of each system is constructed starting from `runID` by appending the following flags:

- **_M** - `MUST` clause is applied to the original query when added to the final `BooleanQuery`.
- **_url_w**X - the system uses URLs support; X refers to the weight applied to the *URLqueries*. If the system also applies GFF and the parameter `excludeURLsFromExpansion` is switched on, then also **NK** is appended.
- **_SYN_w**X - the system uses synonyms in QE; X refers to the weight applied to the synonym in the `SynonymQuery`.
- **_GFF@**K**_w**X - the system uses GFF with K keywords weighted at X.
- **_Q2K@**K**_w**X - the system uses Q2K with K keywords weighted at X.
- **`_reranker@`**Y**`_ScoreOperation_w`**Z - use `reranker` as the re-ranker model for up to the first Y documents retrieved, using as formula `ScoreOperation` with weight Z, if needed.

5.1. Base system & URLs

The first system we developed is the *baseSystem*: in this scenario we do not consider any model or framework in the search phase and only resort to the base first-stage *BM25* retrieval, thus the system performance is purely result of the quality of the Indexing phase. By looking at the Table 2, we can notice that, at this level of complexity, the URLs are not improving the overall performance of the baseline.

From this point on, we will treat the *baseSystem* as a reference point for the systems we are going to describe, so that we can detect the improvements gained by enabling the different techniques during the Searching phase.

Table 2
Overall effectiveness of the base systems with and without the URL support. The best results for each measure are highlighted in boldface, however the use of URLs does not bring any performance changes.

Model	NDCG	MAP	R@100	R@200	R@500	R@1000
baseSystem	**0.47233**	**0.26868**	**0.75861**	**0.82598**	**0.88151**	**0.91686**
url_w10.0	0.47233	0.26868	0.75861	0.82598	0.88151	0.91686
url_w4.0	0.47233	0.26868	0.75861	0.82598	0.88151	0.91686
url_w5.0	0.47233	0.26868	0.75861	0.82598	0.88151	0.91686

5.2. Q2K base: parameters fine-tuning

In order to choose the most suitable base run for Q2K, we focused on the *recall@1000* value. This choice was crucial as in the successive stages, we planned to use reranking on all the 1000 documents retrieved. Therefore, we wanted to favour the runs with higher recall values.

The results can be found in Table 3; analyzing the performances for different number of keywords and keyword weight, we observed a decrease in performance as the number of keywords increased, leading us to opt for a single additional keyword. Additionally, we observed that adding URLs support actually reduces the value of *Recall@1000*.

Based on these findings and on the *Recall@1000* values returned, we concluded that the optimal choice was *Q2K@1_w0.16*, which sets the number of keywords per query to concatenate at 1 and the keyword weight at 0.16. This configuration improves recall compared to other combinations of keywords and weights and, in particular, compared to the base system.

Table 3
Overall effectiveness of the *Q2K* base systems with and without the URL support. The best results for each measure are highlighted in boldface, in particular the focus is on *recall@1000*.

Model	NDCG	MAP	R@100	R@200	R@500	R@1000
baseSystem	**0.47233**	**0.26868**	**0.75861**	0.82598	0.88151	0.91686
M_Q2K@1_w0.16	0.46513	0.26110	0.75599	0.82455	0.88496	0.92045
Q2K@1_w0.12	0.46763	0.26390	0.75732	**0.82689**	0.88528	0.92066
Q2K@1_w0.14	0.46647	0.26198	0.75394	0.82391	0.88480	0.92082
Q2K@1_w0.16	0.46406	0.25906	0.75483	0.82256	**0.88539**	**0.92165**
Q2K@3_w0.12	0.45787	0.25306	0.75054	0.82232	0.88453	0.91782
Q2K@3_w0.16	0.45045	0.24488	0.74120	0.81947	0.88532	0.91733
url_w1.4_Q2K@1_w0.16	0.46750	0.26314	0.75234	0.82300	0.88259	0.91996

5.3. GFF base: parameters fine-tuning

Being GFF one of the main techniques we implemented, we wanted to find the parameters' combination that best improves the *recall@1000*, in order to then improve the overall performance of the system after re-ranking.

First of all, we worked on the number of keywords and their respective weights in the expanded queries to determine the best combination. Then, we tested it on the systems with URLs support to determine their effect when combined with the GFF framework.

The results are presented in Table 4 and we observed the following:

- The combination with higher *recall@1000* was the one with 12 keywords and weight 0.162. In general, we observed that the higher the number of keywords, the better the performance.
- Given the best combination of number of keywords and weights, we were able to slightly increase the *Recall@1000* by using the URLs-supporting system with *URLqueries* weight 1.4.
- Adding the keywords to the *URLqueries* of the expanded queries provided better performance than not adding them, highlighting their importance in URL-matching.

In conclusion, we decided to use systems *GFF@12_w0.162* & *url_w1.4_GFF@12_w0.162* in the next step involving re-ranking, in order to improve the *nDCG*.

Table 4
Overall effectiveness of the *GFF* base systems with and without the URL support. The best results for each measure are highlighted in boldface, in particular the focus is on *recall@1000*.

Model	NDCG	MAP	R@100	R@200	R@500	R@1000
baseSystem	0.47233	0.26868	0.75861	0.82598	0.88151	0.91686
GFF@12_w0.16	0.47512	0.27126	**0.76085**	0.82584	0.88438	0.91792
GFF@12_w0.162	0.47490	0.27128	0.76019	**0.82666**	0.88411	0.91890
GFF@12_w0.1625	0.47478	0.27069	0.75910	0.82441	0.88364	0.91880
GFF@3_w0.162	0.47182	0.26844	0.75803	0.82499	**0.88517**	0.91861
url_w1.4NK_GFF@12_w0.162	0.47467	0.27072	0.76028	0.82478	0.88335	0.91808
url_w1.4_GFF@12_w0.162	0.47497	**0.27130**	0.75959	0.82567	0.88380	**0.91913**
url_w1.5_GFF@12_w0.162	**0.47528**	0.27085	0.75878	0.82635	0.88359	0.91851
url_w1.6_GFF@12_w0.162	0.47477	0.27070	0.75940	0.82588	0.88404	0.91860

5.4. Re-ranker@100: test performance of re-ranker models

One of the many experiments we run was testing our base system against our different re-ranker models and the many score operations on a small subset, i.e. 100, of the documents retrieved by the base system.

The goal of these experiments was to get a general intuition on the best performing re-ranker models to be used in our final systems; Additionally, we were also interested in proving the strength of the GFF framework when presented with a cross-encoder for re-ranking.

In particular, we observed how Recall & Precision changed under the threshold of 100 documents, as well as changes in the *nDCG*.

The results are presented in Table 5; the most important resulting observations are the following:

1. The bi-encoder approaches, SBERTAVERAGE & SBERTSUM, degrade performances by a significant margin in all the metrics compared to the base system, even with GFF; the most effective document-to-sentences-to-document embedding aggregation technique is *ADD*, while the score operation is ADD. This behaviour calls for more in-depth studies on its origin, raising the following research points:
    - Different models may be needed and fine-tuning may prove successful.

- Bi-encoders may need larger documents pool to prove effective.
- Multilingual bi-encoder models might not generalize well to mainly monolingual documents.
- Bi-encoders may not be as effective as `BM25Similarity` in scoring web pages, maybe due to their noisy nature.
- Alternative re-scoring formulas might be needed, to better capture the similarity between queries and documents when applied on top of, or instead of, first-stage `BM25Similarity` retrieval.

2. As expected, the cross-encoder approach to re-ranking with MMARCO proves really successful, even on the small subset of 100 documents. This confirms their positive effects on last-stage ranking and their increased performance compared to bi-encoders[6].
3. The best value for the max length of the query-sentence pairs tokens is 128.
4. On the cross-encoder, the ADD score operation at weight 5 seems to perform better overall than the `MAXMIN_ADD`. However, the performance difference is small and the `MAXMIN_ADD` performs better in some cases, like the GFF system shows. Given the small pool of documents re-ranked, we decided to keep both the operations in our final runs, since the two may perform differently when presented with different conditions.
5. As expected, the GFF framework significantly increases the performance of cross-encoders and, given the small pool of documents the re-ranking has been applied on, we expect to observe even better performances when that pool is increased.

On a final note, we specify that the *GFF* systems have been run with parameters:

- `GFFLeftoverDocumentsLimit` $= 100$,
- `GFFMinimumDocumentsToRerank` $= 100$.

Table 5
Overall effectiveness of our re-ranker models on a small portion of the retrieved documents. The best results for each measure are highlighted in boldface, in particular the focus is on *nDCG* and the changes in *Precision-Recall* brought by re-ranking.

Model	NDCG	MAP	P@5	P@10	P@50	P@100	R@50	R@100
baseSystem	0.47233	0.26868	0.26711	0.22387	0.09653	0.05549	0.66408	0.75861
128GFF@12_w0.162_MMARCO@100_ADD_w5	0.49927	0.30340	**0.30250**	0.25075	0.10167	0.05674	0.70024	0.77650
128GFF@12_w0.162_MMARCO@100_MAXMIN_ADD_w5	**0.50586**	**0.31018**	0.30150	**0.25643**	**0.10371**	**0.05895**	**0.71572**	**0.80665**
128MMARCO@100_ADD_w5	0.49906	0.30496	0.30017	0.25559	0.10080	0.05549	0.69325	0.75861
96MMARCO@100_ADD_w5	0.49673	0.30123	0.29716	0.25225	0.10030	0.05549	0.69064	0.75861
64MMARCO@100_ADD_w5	0.49654	0.29962	0.29716	0.24691	0.09990	0.05549	0.68937	0.75861
MMARCO@100_MAXMIN_ADD_w5	0.49709	0.30254	0.29783	0.25426	0.10070	0.05549	0.69259	0.75861
SBERTAVERAGE@100_ADD_w5	0.45169	0.24816	0.23973	0.21135	0.09452	0.05549	0.65156	0.75861
SBERTAVERAGE@100_MAXMIN_ADD_w5	0.44806	0.24269	0.23472	0.20601	0.09459	0.05549	0.65270	0.75861
SBERTSUM@100_ADD_w5	0.45612	0.25050	0.24341	0.20417	0.09279	0.05549	0.64173	0.75861
SBERTSUM@100_MAXMIN_ADD_w5	0.45044	0.24258	0.23472	0.20217	0.09302	0.05549	0.64368	0.75861
GFF@12_w0.162_SBERTSUM@100_ADD_w5	0.45629	0.25032	0.24674	0.20484	0.09249	0.05514	0.63921	0.75509

5.5. Re-ranker@1000: final systems with MMARCO@1000

To produce our final systems, we applied re-ranking to all the 1000 retrieved documents, so that we could analyze which system produces the best performance in the *nDCG* metric.

As described in the previous section, we chose the MMARCO cross-encoder approach with max length of the query-sentence pairs tokens at 128; we use both ADD and `MAXMIN_ADD` as score operation to see if, on a bigger set of documents, their results were still similar.

Starting from these configurations, our goal was to show the improvement of *nDCG* at a higher number of re-ranked documents on our base system, on the GFF framework and when applying Q2K technique for QE.

Since we observed that the use of the URLs combined with GFF led us to small improvements of the overall performance, we decided to also use a model with score operation ADD in order to try to obtain further improvements in performance.

The results of our final systems are presented in Table 6.

By comparing the systems we observe that:

1. As we have seen for the re-ranking@100, the difference between the ADD score operation at weight 5 and the MAXMIN_ADD is not significant. Depending on whether the system involves Q2K or other techniques, the score operations may lead to better or worse results with respect to the other.
2. The re-ranking on all the set of retrieved documents led the systems to gain additional overall performance.
3. The use of the URLs-supporting GFF system improves the quality of ranking since it moves in a higher position documents that are relevant.

Table 6
Overall effectiveness of the *Re-ranker@1000* with *MMARCO@1000* systems with and without the URL support. The best results for each measure are highlighted in boldface, for these systems the focus is on the overall measures in particular on *nDCG*.

Model	NDCG	MAP	P@5	P@10	R@200	R@1000
baseSystem	0.47233	0.26868	0.26711	0.22387	0.82598	0.91686
GFF@12_w0.162_MMARCO@1000_ADD_w5	0.50897	0.31311	0.30083	0.25893	**0.86591**	0.91978
GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5	0.50677	0.31073	0.30217	0.25659	0.86358	0.91916
MMARCO@1000_ADD_w5	0.50812	0.31321	**0.30484**	**0.25977**	0.86294	0.91686
MMARCO@1000_MAXMIN_ADD_w5	0.50568	0.30980	0.30184	0.25860	0.86432	0.91686
Q2K@1_w0.16_MMARCO@1000_ADD_w5	0.50582	0.31030	0.29850	0.25760	0.86391	0.91930
Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5	0.50612	0.30892	0.30050	0.25710	0.86471	**0.92082**
url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5	**0.50906**	**0.31325**	0.30250	0.25860	0.86441	0.91956

6. Results and Discussion

In this section, we provide a discussion on the results of our systems and their overall performance, with a particular focus on the temporal evolution across the training January collection and the two test collections of June and August.

Considering the results discussed in Section 5, our 5 final submitted systems to Task 1 - LongEval-Retrieval of the LongEval CLEF 2024 Lab are the following:

1. url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5,
2. GFF@12_w0.162_MMARCO@1000_ADD_w5,
3. GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5,
4. MMARCO@1000_ADD_w5,
5. Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5.

For the construction of system IDs and the absolute performance measures of the systems on the January training data, refer to Section 5.

6.1. Results on test data

The LongEval-Retrieval Lab provided us with the results of our systems on the test sets of June and August, Table 7 reports them and allows us to follow with some considerations.

First of all, we notice that almost all systems suffered a loss of approximately 0.1 in *nDCG* and 0.07 in *MAP* across the time lags; this behaviour is coherent with the performance degradation over time of web IR systems and will be the subject of deeper analysis going forward in this work.

Another interesting observation that arises is the change in "ranking" of our systems, focusing on *nDCG*, we can observe the following:

- June: the ranking does not change much, except for system 3, *GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5*, slightly outperforming system 4, *MMARCO@1000_ADD_w5*.
- August: the ranking between systems shows a noteworthy change, even though still being slightly apart from each other; this might be an indicator of the need to update the systems, since the temporal evolution of the collection seems to "flatten" the differences between systems over time.

Table 7
Effectiveness of the final systems on both test sets. The best results for each measure are highlighted in boldface, for these systems the focus is on the *nDCG* values for June and August.

#	Model	NDCG June	NDCG August	MAP June	MAP August
1	url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5	**0.39229**	0.29340	**0.24401**	0.17062
2	GFF@12_w0.162_MMARCO@1000_ADD_w5	0.39222	**0.29359**	0.24391	0.17063
3	GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5	0.39188	0.29339	0.24207	0.17041
4	MMARCO@1000_ADD_w5	0.39131	0.29326	0.24325	0.17056
5	Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5	0.39022	0.29356	0.23984	**0.17077**

6.2. Overall performance over time

Given all the pieces collected so far, we can now discuss the results of our systems in depth. Figure 5 presents the performance of our final systems on the 3 provided time lags in the form of box plots.

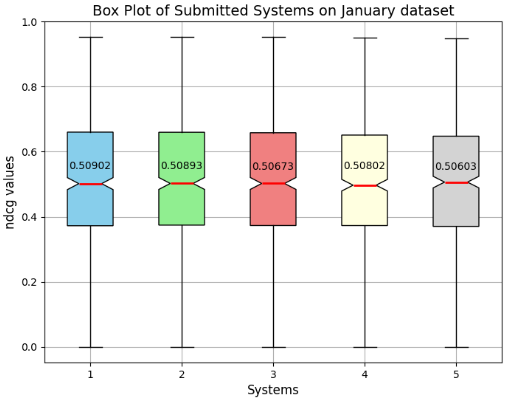
First of all, we want to discuss the effect of temporal evolution of the collection that emerges when comparing the box plots of our systems across the 3 time lags. In particular, we focus this discussion on the *nDCG* values.

The deterioration in performance of our systems over time can be attributed to the evolution of the collection across time lags: the more time passes, the more "new" queries appear for which our systems are unable to consistently retrieve relevant documents and rank them highly. This idea is supported by the following observations:

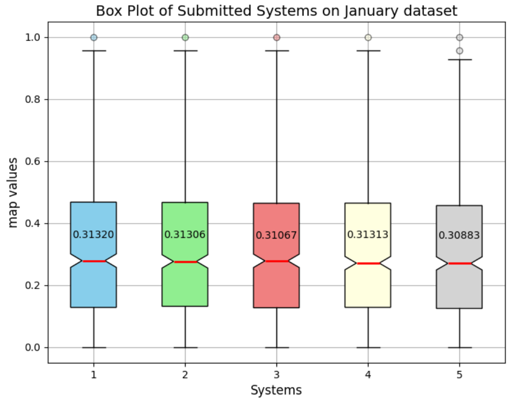
- Across time lags, the mean performance values decrease significantly and their distribution across the queries are progressively more skewed to lower values; additionally, the presence of outliers increases as time passes.
- Looking at the evolution of interquartile ranges among time lags, we observe that at January and August the distribution of *nDCG* values across queries are focused around the mean, while at June the distribution suffers a greater dispersion.

These observations imply that, as time passes, our systems are still able to perform well for certain queries, but their number decreases significantly over time, indicating the presence of queries for which we do not have relevant-enough documents to retrieve. In particular, we can observe that:

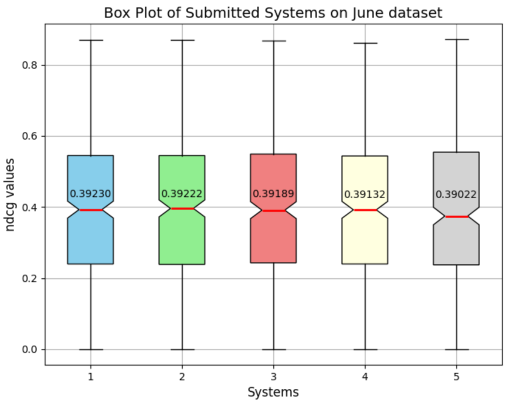
1. At January, the performance is good and consistent: for each query we retrieve relevant documents and rank them highly.
2. At June, the performance suffers a significant overall loss and a greater dispersion across queries occur: now for a portion of queries the performance becomes poor and, for another portion, it keeps being as good as January, suggesting a sort of balance between "new" poor-performing queries and well-performing ones.
3. At August, the overall performance deteriorates significantly and the distribution focuses around low *nDCG* values across most queries: now the number of "new" queries for which the systems do not perform well increases significantly and dominates the mean performances, leaving a few well-performing outliers.



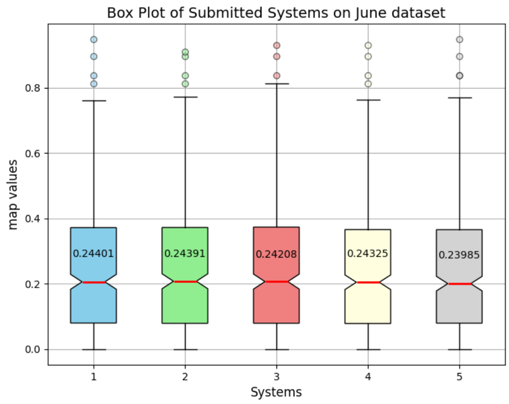
(a) Final systems nDCG performance on January train set.



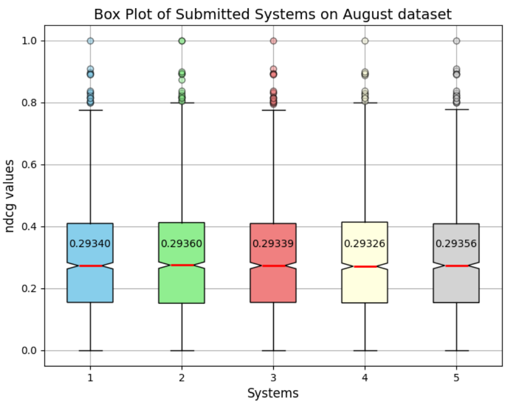
(b) Final systems MAP performance on January train set.



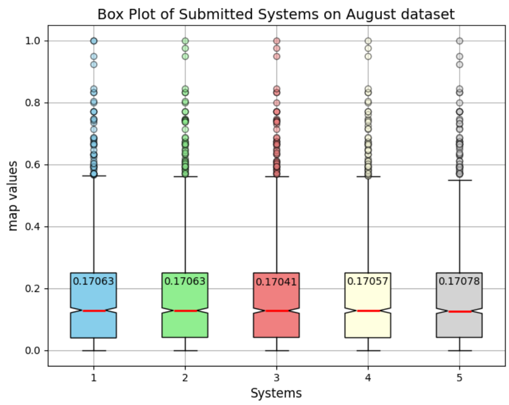
(c) Final systems nDCG performance on June test set.



(d) Final systems MAP performance on June test set.



(e) Final systems nDCG performance on August test set.



(f) Final systems MAP performance on August test set.

Figure 5: Report on overall performance of our final systems on the January train set, June test set and August test set. For the system ID, refer to the numbers associated to the submitted runs discussed in section 5.5.

Finally, looking at the overall results, we want to spend some words on the need for updates of the systems over time: while at June it can be argued that the overall performance is still "good enough", at August the performance has deteriorated to such a point that an update of the system is inevitable; in particular, we suggest that good indicators for the need to update the systems can be found in the relation between interquartile range and mean performance values, since they allow to control the overall performance and its dispersion across queries.

6.3. Temporal performance evolution: nDCG drop analysis

Since our final objective is developing systems resilient to the temporal evolution of the collection, in this section we want to dedicate some time to analyse the effects of our techniques on the *nDCG* drop between time lags, if any effect is observable.

In order to measure the temporal resilience of our systems, we decided to compute the *"drop-percentage"* for each time lag from the train set results, i.e.:

$$\text{drop}_{\%} = 100\frac{\text{nDCG}_{\text{lag}}}{\text{nDCG}_{\text{train}}}$$

Table 8
nDCG percentage drop of our final and corresponding base systems from the January runs to the two time lags (June and August).

#	Model	January-June drop%	January-August drop%
1	url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5	77.06164	57.63564
2	GFF@12_w0.162_MMARCO@1000_ADD_w5	77.06151	57.68512
3	GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5	77.33093	57.89411
4	MMARCO@1000_ADD_w5	77.01133	57.72848
5	Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5	77.10029	58.00205
0	baseSystem	78.71615	57.09990
0	GFF@12_w0.162	78.20593	57.08570
0	Q2K@1_w0.16	79.10615	57.83734
0	url_w1.4_GFF@12_w0.162	78.36284	57.11939

Table 8 presents the results on our system and highlights the following:

- The dynamic `MAXMIN_ADD` score operation on re-ranking better mitigates the performance loss compared to the fixed weight `ADD` operation. This can be observed both in June and August and is consistent with our intuition on the two score operations.
- Re-ranking does not seem to be effective in mitigating performance drops over time and, in general, it appears to be affected by a relative performance drop proportional to the size of the interquartile range, i.e. dispersion of *nDCG* values across queries, at the respective time lag when compared to systems that do not implement re-ranking. This observation seems intuitively coherent with the relation and interaction that re-ranking may have with the first-stage retrieved documents for each query when a "greater dispersion of performance" occurs.
  In other words: at January the majority of queries are "well-performing", i.e. the systems retrieve a significant portion of the relevant documents and rank them high, and thus the improvement in performance produced by re-ranking is high when compared to the performance of base systems. At June instead, the bigger interquartile ranges suggest a mixed presence of "well" and "poor" performing queries, leading to a bigger drop of performance for re-ranking systems (since they "start higher") compared to the drop of base systems. Finally, at August, the contained interquartile ranges and low mean performance values suggest the presence of mainly "poor-performing" queries, thus the effect of re-ranking grants a lower relative performance drop than base systems. However, a future study on more time lags and different models is needed to highlight more definitive results and deeper conclusions.

In conclusion, ad-hoc techniques like dynamic weight re-ranking formulas seems to be successful in mitigating performance losses by a small amount; however, further research on other ad-hoc techniques to significantly increase resilience is needed, if any significant improvement can be achieved at all.

6.4. Topics evolution over time

In order to further investigate the role played by the evolution of queries over time in the performance drop of IR systems, we compute the Jaccard distance between the sets of queries at the three time lags.

In particular, before computing the distance between each set, we applied the Analyzer component to the queries, in order to get the tokens our systems are actually searching for.

The results, presented in Table 9, reinforce our hypothesis about the appearance of "new" queries over time, since the more time passes, the greater the Jaccard distance between time lags grows.

Table 9
Jaccard distances between the set of queries at the three time lags.

Time lags	Jaccard Distance
January - June	0.71300
June - August	0.75812
January - August	0.76465

This may be motivated by a shift in the users research interests over time and, again, highlights the need for system updates in order to better answer the user information needs.

7. Statistical analysis

In this chapter we want to wrap-up the discussion we presented so far by performing statistical significance tests to see whether we can reject the null hypothesis between QE techniques and "base" systems, between "base systems" and "re-ranking systems" and, finally, between our final submitted systems.

The statistical tools we utilize in this chapter are the two-way *ANalysis Of VAriance (ANOVA)* and Multiple Comparisons with Tukey's HSD test, applied only on the *nDCG* values. For all tests we have chosen a significance level $\alpha = 0.05$.

7.1. Analysis of statistical significance of QE techniques

In this section, our goal is to study whether the use of QE techniques has a statistically significant effect on the systems. The systems we study in this section are the "base version", without re-ranking, of the QE techniques we use in our final submitted systems.

Results on the two-way ANOVA are presented in table 10 and, considering also the multiple comparisons between the base QE systems reported in Figure 6, we conclude the following:

- Train (January): the ANOVA test allows us to reject the null hypothesis, since $p < \alpha$; this tells us that there is a statistically significant difference between the different QE techniques, as well as using them or not.
  Looking at the multiple comparisons, we indeed observe that the lower performance of the Q2K is statistically significant, proving again our Q2K implementation to be the less effective between our final techniques, being actually a little detrimental to *nDCG* performance compared to the base system.
  Another important observation is that the improvement brought by the use of GFF techniques proves to be too contained to be statistically significant compared to the base system; however, we argue that, even though not statistically significant, the improvement introduced by GFF may be practically significant, since it may lead to better user satisfaction.
- Test (June and August): the ANOVA test fails to reject the null hypothesis, since $p > \alpha$. In particular, the results appear to be coherent with the discussion of Section 6.2 on the passage of time, since the increasing p-values from ANOVA and the multiple comparisons allow us to see again the flattening effect on the *nDCG* performance distributions that time has on the systems.

Table 10
Two-way ANOVA on the "base systems" (no re-ranking) to test statistical significance of QE techniques across the January (a), June (b) and August (c) time lags.

(a) Two-Way ANOVA performed on the base systems on the January training set.

Source	SS	df	MS	F	Prob>F
Systems	0.0477	3	0.0159	13.2247	1.5343e-08
Topics	93.7170	598	0.1567	130.3901	0
Error	2.1562	1794	0.0012	[]	[]
Total	95.9209	2395	[]	[]	[]

(b) Two-Way ANOVA performed on the base systems on the June test set.

Source	SS	df	MS	F	Prob>F
Systems	0.0068	3	0.0023	1.6456	0.1771
Topics	67.0996	403	0.1665	121.3346	0
Error	1.6590	1209	0.0014	[]	[]
Total	68.7654	1615	[]	[]	[]

(c) Two-Way ANOVA performed on the base systems on the August test set.

Source	SS	df	MS	F	Prob>F
Systems	0.0081	3	0.0027	0.2034	0.8941
Topics	131.7012	1517	0.0868	6.5358	0
Error	60.4519	4551	0.0133	[]	[]
Total	192.1612	6071	[]	[]	[]

7.2. Analysis of statistical significance of re-ranking

In this section, we aim to analyse the statistical significance of improvements introduced by the use of re-ranking. In particular, we run our hypothesis testing against a set of 9 systems consisting of the 5 submitted final systems and their respective 4 base systems, i.e. without re-ranking.

First of all, looking at Table 11, the ANOVA test allows us to reject the null hypothesis across all the time lags, since we obtain $p < \alpha$ in each situation, proving the central importance of re-ranking in IR systems.

Since the ANOVA test confirmed that at least one among the systems is different from the others, we now apply the Tukey's HSD test to verify the pairwise statistical significance of the difference between re-ranking systems and base systems. The results are presented in Figure 7.

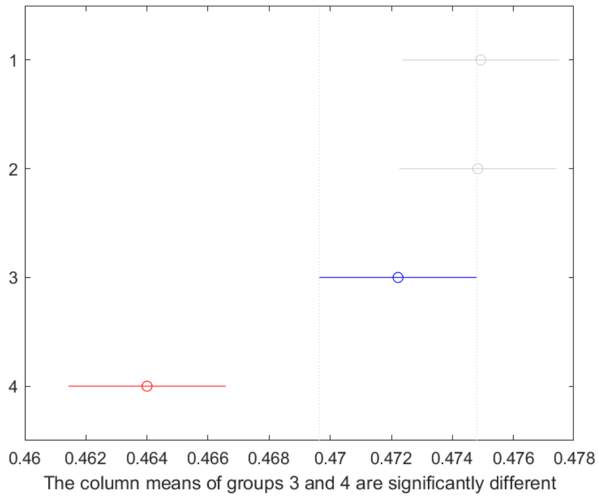
Re-ranking proves to be a crucial technique to make an effective IR system, since it introduces a statistically significant improvement even across time lags.

Additionally, it is interesting to observe how re-ranking dominates the performance of systems, since a greater overlap between the re-ranking group can be observed compared to the "base" group across all time lags.

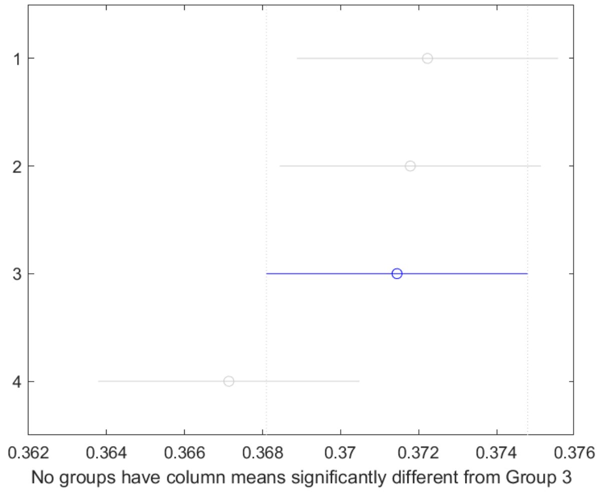
On a side note, the overall behaviour appears to be coherent with the temporal evolution discussion so far: when the systems are provided with "new" queries for which they are not able to retrieve relevant-enough documents, the improvement of GFF techniques falls off over time, since the more time passes, the less relevant documents there are for each query and the less the increased reach over documents of GFF matters, in particular when the collection of retrieved documents goes through a re-ranking phase.

7.3. Analysis of statistical significance on final systems across time-lags

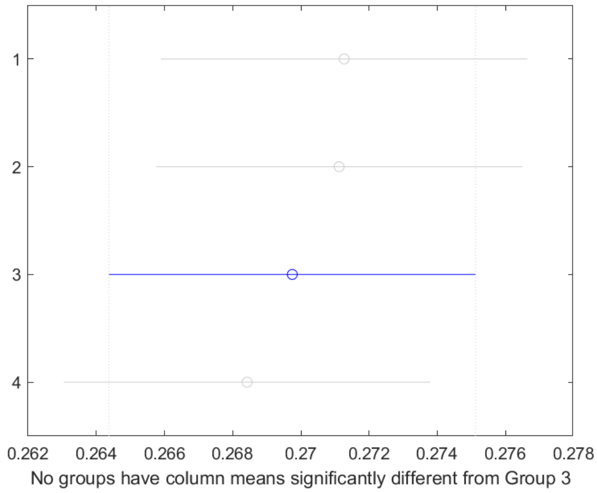
Finally, we want to test whether there is a statistically significant difference between our final submitted systems.



(a) QE techniques multiple comparison on January.



(b) QE techniques multiple comparison on June.



(c) QE techniques multiple comparison on August.

#	System ID
1	url_w1.4_GFF@12_w0.162
2	GFF@12_w0.162
3	BaseSystem
4	Q2K@1_0.16

(d) Legend for figure a, b and c.

Figure 6: Multiple comparison with Tukey's HSD test on the 3 time lags with the base system & QE techniques, with no-reranker. The aim of this test is to check statistical significance

First of all, Table 12 presents the results of two-way ANOVA test on the 5 final systems and, since $p > \alpha$ for all time lags, we fail to reject the null hypothesis. Additionally, looking at the p-values over time, we find again a behaviour coherent with the temporal evolution discussion of Section 6.2. This behaviour is also strengthened by the multiple comparisons with Tukey's HSD test on the final systems presented in Figure 8: we again fail to reject the null hypothesis and, as time passes, we can observe the same behaviour discussed in Section 6.2.

Another interesting aspect emerges again from comparing results with Section 7.1: since we now fail to reject the null hypothesis between final systems, and taking into consideration the increased overlap between them (compared to the "base" systems), we can observe again how re-ranking dominates the performances over QE techniques.

On a final note, we want to add some considerations about failing to reject the null hypothesis between our final systems: as already argued in Section 7.1, the small improvements introduce by GFF, even though statistically not significant, may prove practically significant since they are consistent and may result on a better user-experience.

Additionally, the discussion opens an interesting future research topic regarding the GFF technique and the trade-off between performance and computation time: considering that the use of GFF increases the computational time proportional to the increase in size of the total pool of documents retrieved per query, how does the performance-time trade-off scale when limiting the retrieval per expanded

Table 11
Two-Way ANOVA on the base systems and final systems to test statistical significance of our re-ranking technique across January (a), June (b) and August (c).

(a) Two-Way ANOVA performed on the final systems and baseline on the training set.

Source	SS	df	MS	F	Prob>F
Systems	1.8013	8	0.2252	67.3619	3.1250e-105
Topics	197.0354	598	0.3295	98.5764	0
Error	15.9905	4784	0.0033	[]	[]
Total	214.8271	5390	[]	[]	[]

(b) Two-Way ANOVA performed on the final systems and baseline on lag6.

Source	SS	df	MS	F	Prob>F
Systems	0.4017	8	0.0502	17.1105	3.8379e-25
Topics	145.4552	403	0.3609	122.9979	0
Error	9.4607	3224	0.0029	[]	[]
Total	155.3175	3635	[]	[]	[]

(c) Two-Way ANOVA performed on the final systems at lag8.

Source	SS	df	MS	F	Prob>F
Systems	1.8466	8	0.2308	19.0019	1.1516e-28
Topics	309.4818	1517	0.2040	16.7941	0
Error	147.4240	12136	0.0121	[]	[]
Total	458.7524	13661	[]	[]	[]

Table 12
Two-way ANOVA on the final submitted system across the January (a), June (b) and August (c) time lags.

(a) Two-Way ANOVA performed on the final systems on the training set.

Source	SS	df	MS	F	Prob>F
Systems	0.0043	4	0.0011	1.5803	0.1768
Topics	115.5409	598	0.1932	286.7509	0
Error	1.6117	2392	6.7380e-04	[]	[]
Total	117.1569	2994	[]	[]	[]

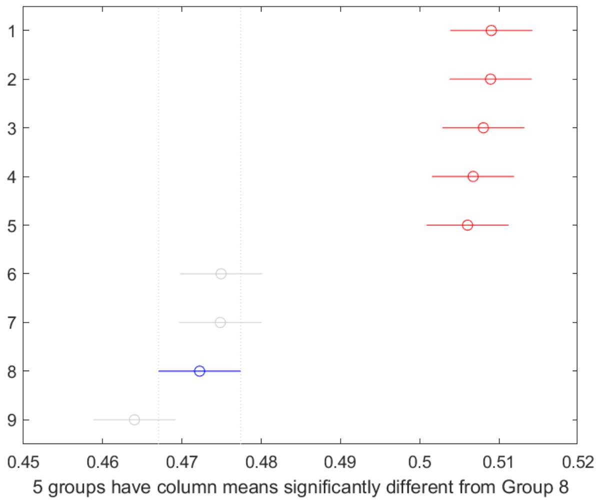
(b) Two-Way ANOVA performed on the final systems on the June test set.

Source	SS	df	MS	F	Prob>F
Systems	0.0012	4	2.9726e-04	0.6217	0.6471
Topics	85.3864	403	0.2119	443.1389	0
Error	0.7707	1612	4.7813e-04	[]	[]
Total	86.1584	2019	[]	[]	[]

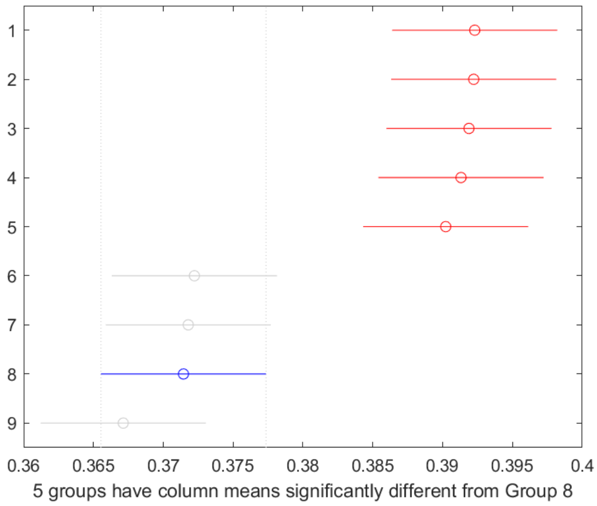
(c) Two-Way ANOVA performed on the final systems on the August test set.

Source	SS	df	MS	F	Prob>F
Systems	5.2976e-05	4	1.3244e-05	0.0011	1
Topics	194.7365	1517	0.1284	0.1284	0
Error	70.0161	6068	0.0115	[]	[]
Total	264.7527	7589	[]	[]	[]

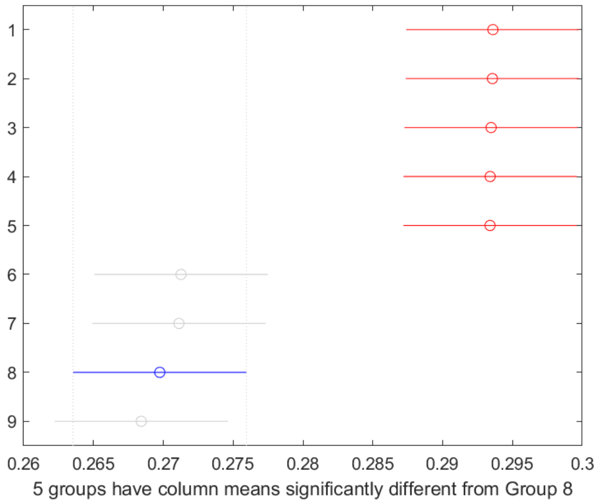
queries to a smaller total pool of documents (i.e., including already-retrieved documents) than the one we used in this work? Since the technique proved to be effective in achieving consistent small improvements, we expect it to provide better relative performance improvements at a significantly lower computation time when limiting the re-ranking phase to smaller total sets of documents per query, in particular if we consider the empirical observation on the low quality of the "last-ranked" documents retrieved by the expanded queries. In other words, the research question can be summarized



(a) Multiple comparisons on January train set.



(b) Multiple comparisons on June test set.



(c) Multiple comparisons on August test set.

#	System ID
1	url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5
2	GFF@12_w0.162_MMARCO@1000_ADD_w5
3	GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5
4	MMARCO@1000_ADD_w5
5	Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5
6	url_w1.4_GFF@12_w0.162
7	GFF@12_w0.162
8	BaseSystem
9	Q2K@1_0.16

(d) Legend for figure a, b and c.

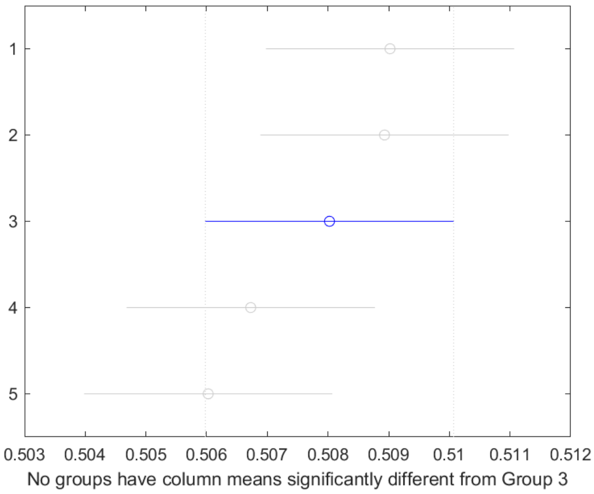
Figure 7: Multiple comparison with Tukey's HSD test on the 3 time lags between the base systems and the re-ranker implementing final systems.

as: does the GFF technique introduces more incisive and statistically significant improvements when limiting the number of retrieved/re-ranked documents and does the overall performance-time trade-off improves significantly?

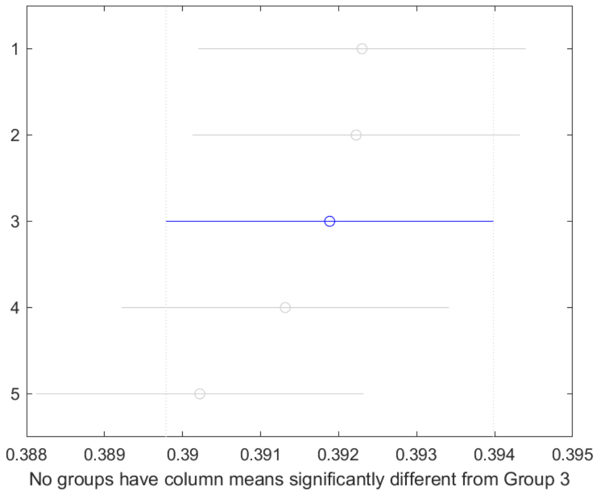
8. Conclusions

In this work, we prove the effectiveness of cross-encoders as final-stage re-rankers, as well as the potential strength of the GFF technique both stand-alone for improving QE and as a noise reduction system to improve cross-encoders performance. Another point of interest that emerges from our work is the observation that leveraging URLs information does not bring increased performance per-se but, when coupled with noise reduction systems like GFF, it can bring to better performances.

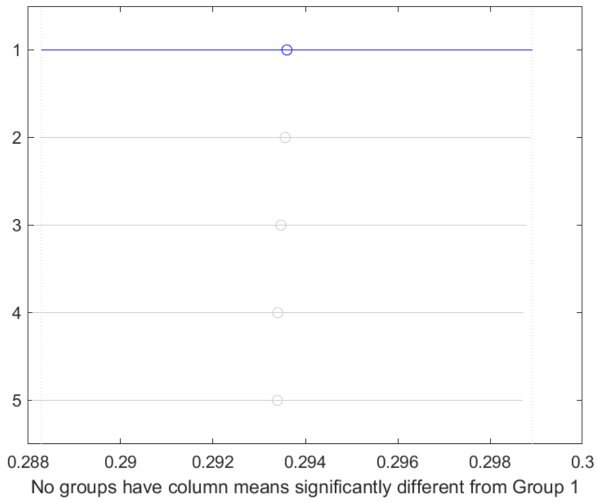
Our experiments on re-ranking@100 with bi-encoder models prove them ineffective as a re-ranking technique applied on top of *BM25* first-stage retrieval for web search. Additionally, our final systems do not present statistically significant differences between them. Those finding, combined with the research points raised in Section 5.4 and 7.3, give us the idea to explore multi-stage GFF retrieval systems that leverage *BM25* to get a significantly larger pool of documents to be then re-ranked with a fast document-to-token bi-encoder approach using the *BERTScore* [14] formula and a final last-stage



(a) Final systems multiple comparison on January.



(b) Final systems multiple comparison on June.



(c) Final systems multiple comparison on August.

#	System ID
1	url_w1.4_GFF@12_w0.162_MMARCO@1000_ADD_w5
2	GFF@12_w0.162_MMARCO@1000_ADD_w5
3	GFF@12_w0.162_MMARCO@1000_MAXMIN_ADD_w5
4	MMARCO@1000_ADD_w5
5	Q2K@1_w0.16_MMARCO@1000_MAXMIN_ADD_w5

(d) Legend for figure a, b and c.

Figure 8: Multiple comparison with Tukey's HSD test on the 3 time lags for the final systems. The numbers refer to the submitted systems, as reported in the "Sumbmitted runs" paragraph of Section 5.5.

cross-encoder re-ranker on a smaller pool of documents.

As future work, we also want to further explore the differences between ADD and MAXMIN_ADD re-scoring formulas, as well as new possible solutions, in particular around *BERTScore* [14].

Finally, on the temporal evolution side, our work highlights again the well-known performance deterioration effect caused by the passage of time and provides insights onto the temporal evolution of the techniques explored in our systems.

References

[1] M. Li, H. Zhuang, K. Hui, Z. Qin, J. Lin, R. Jagerman, X. Wang, M. Bendersky, Generate, filter, and fuse: Query expansion via multi-step keyword generation for zero-shot neural rankers, 2023. arXiv:2311.09175.

[2] 7digital, Synonym list, https://github.com/7digital/synonym-list/tree/master/, 2014.

[3] The Apache Software Foundation, https://lucene.apache.org/, 2024.

[4] N. Reimers, I. Gurevych, Making monolingual sentence embeddings multilingual using knowledge distillation, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2020. URL: https://arxiv.org/abs/2004.09813.

[5] Nils Reimers, nreimers/mmarco-mminilmv2-l12-h384-v1, https://huggingface.co/nreimers/mmarco-mMiniLMv2-L12-H384-v1, 2024. Accessed: 2024-05-01.
[6] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019. URL: http://arxiv.org/abs/1908.10084.
[7] The Apache Software Foundation, Apache opennlp french sentence detection model, https://opennlp.apache.org/models.html, 2024. Accessed: 2024-05-01.
[8] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, W. E. Sayed, Mistral 7b, 2023. `arXiv:2310.06825`.
[9] G. V. Cormack, C. L. A. Clarke, S. Buettcher, Reciprocal rank fusion outperforms condorcet and individual rank learning methods, in: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 758–759. URL: https://doi.org/10.1145/1571941.1572114. doi:`10.1145/1571941.1572114`.
[10] LongEval, LongEval 2024 Train Collection, https://researchdata.tuwien.at/records/y60e9-k9b51, 2024.
[11] LongEval, LongEval 2024 Test Collection, https://researchdata.tuwien.ac.at/records/xr350-79683, 2024.
[12] NIST, trec_eval, https://github.com/usnistgov/trec_eval, 2019.
[13] E. Bassani, ranx: A blazing-fast python library for ranking evaluation and comparison, in: ECIR (2), volume 13186 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 259–264. doi:`10.1007/978-3-030-99739-7\_30`.
[14] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, Y. Artzi, Bertscore: Evaluating text generation with bert, 2020. `arXiv:1904.09675`.
[15] LongEval, LongEval 2024, https://clef-longeval.github.io/, 2024.

A. Description of the Parser component for noise elimination

Our systems implement a `JSONParser` class, designed to handle content cleaning of documents, such as removing emojis, URLs, HTML, and code, based on various flags specified in the configuration file `ParserParams.xml`.

Once the cleaning is completed, the document is converted into an instance of `ParsedDocument`, that contains two fields: an "ID" field representing the document identifier, a "BODY" field containing the document body. Furthermore, we have also included an additional optional field "URL" intended to store the web address of the document, that could be useful especially in the case of queries directly involving website URLs.

A.1. Parser Flags Description

Within the configuration file, there is a wide range of flags used to manage the cleaning process, offering control over the elimination of noise from documents. The list of flags, along with their respective functions, is the following:

- **clearEmojis**: Controls whether emojis should be removed from the document.
- **clearHTML**: Controls whether HTML tags should be removed from the document.
- **clearURLs**: Controls whether URLs should be split into their components and the non-informative URL syntax removed from the document contents.
- **clearCode**: Controls whether code blocks should be removed from the document.
- **clearTrailingPunctuation**: Controls whether trailing punctuation should be removed from the document.

- **clearNonStandardPunctuation**: Controls whether non-standard punctuation should be removed from the document.
- **clearTelephone**: Controls whether telephone numbers should be removed from the document.

All the flags are assumed to be set to on in our systems.

B. Description of the Analyzer component for text processing

Our system implements two distinct analyzers, that allow us to handle both processing of French documents and English ones, in order to have a specific behaviour for each collection.

Their behaviour can be controlled by specifying the appropriate parameters in the relative `[Language]AnalyzerParams.xml` file.

B.1. French analyzer

This analyzer is crafted to handle French collections and it accepts the following parameters:

- **Tokenizer**: determines how the text is divided into tokens. Our project implements the following alternatives:
  - **StandardTokenizer**: uses advanced rules, defined by the Unicode Text Segmentation algorithm, to segment text into tokens.
  - **WhiteSpaceTokenizer**: Splits text into tokens based on whitespace characters.
  - **LetterTokenizer**: Breaks text into terms whenever it encounters a character which is not a letter.
- **Filters**: change tokens by applying rules, and remove the ones that do not respect particular conditions.
  - **Lower Case Filter**: Converts tokens to lowercase;
  - **Minimum Length**: Specifies the minimum allowed length of each token;
  - **Maximum Length**: Specifies the maximum allowed length of each token.
  - **English stopList**: removes common English words that are not meaningful for the content of the document. It is also applied to the French Analyzer since, during the experimental phase, we observed that it leads the systems to have an improvement of the performance, in particular for those that implement our QE techniques. We suggest that this behaviour is caused by the presence of English words in the documents that, after being stemmed by the French stemmer, correspond to stems of French words with potentially different meanings; however, this behaviour still needs further analysis and calls for future research work.
  - **ElisionFilter**: it targets elisions, so it removes articles, prepositions and conjunctions, hyphens, apostrophes.
  - **French stopList**: serves the same purpose as the English stoplist, but it works for French words.
- **Stemming filters**: reduce words to their root form (stem), enhancing search recall by considering word variations.
  - **Snowball Filter**: uses the Snowball stemming algorithm for French.
  - **FrenchLightStemmer**: this filter offers greater flexibility and adaptability in managing various forms of French language text.

Our experiments showed us that the different tokenizers have roughly the same performances, in this situation we opted to use the `StandardTokenizer`, since it achieved slightly better performances and a more accurate tokenization. As filters we used the *LowerCaseFilter* and *ElisionFilter*, additionally, we set the *minimumLength* at 2 and the *maximumLength* at 15, since a lower number was too restrictive for our queries. Finally, for stemming, we used the *FrenchLightStemmer*.

We tried different combinations with the other possible settings, but in our case we saw that the performances were worse than the one just described.

B.2. English analyzer

The English analyzer performs the same logical steps as the French one, but it is specialized for the English collection. It allows the following parameters:

- **Tokenizer**: determines how the text is divided into tokens. The English analyzer implements the same tokenizers described for the French analyzer in Section **??**.
- **Filters**:
  - **Lower Case Filter**: Converts tokens to lowercase;
  - **Minimum Length**: Specifies the minimum allowed length of each token;
  - **Maximum Length**: Specifies the maximum allowed length of each token.
  - **English stopList**: removes common English stopwords.
  - **English Possessive Filter**: determines whether the English possessive filter should be applied.
  - **synonymGraphFilter**: for each token it maps single (or multi) token synonyms, producing a fully correct graph output. If used for indexing it must be followed by FlattenGraphFilter to squash tokens on top of one another, because the indexer cannot directly consume a graph.
- **Stemming filters**:
  - **Snowball Filter**: uses the Snowball stemming algorithm for English.
  - **English Minimal Stem Filter**: Applies a minimal plural stemmer for English;
  - **K Stem Filter**: Implements the K stemming algorithm.

In our implementation, we tested different combinations of parameters to efficiently preprocess text before indexing and querying; however, as already mentioned, the performances of English systems proved to be significantly lower than French ones, so we suggest again to focus on the latter.

On a final note, another test scenario involved the use of the `SynonymGraphFilter` with the synonym collection `engSynonyms.txt`, which contains pairs of words representing British and American versions of the same concept. This synonym collection was obtained from the `synonym-list` repository on GitHub [2]. Employing this collection could have been useful to ensure greater uniformity of results, making the query independent from the linguistic variant used by the user. However, while it slightly improves the English system's performance, its performances decrease when applied alongside query expansion techniques, rendering the filter marginal or nearly redundant.

C. Prompts used for keyword generation through LLMs in French and English

Table 13 and 14 report respectively the French and English prompt provided to the mistralai/Mistral-7B-Instruct-v0.2 LLM model to generate the keywords used in the query expansion phases of our systems.

Table 13
Few-shots prompt used for generating keywords related to a provided user query in French.

```
You are a bot used for query expansion.
Your task is to provide a list of 10 keywords, strictly in french, that might be related to a french query
provided after <<<>>>.

You must respond in a single line with the list of 10 keywords in csv format inside [[]].
Do not include the word "KEYWORDS".
Do not provide explanations or notes.
Do not provide translations or other additional information beside the keywords.

###
Here are some examples:

QUERY: lequel des éléments suivants est le principal facteur de risque du cancer du col de l'utérus?
KEYWORDS: [[ HPV, papillomavirus, système immunitaire, souches ]]

QUERY: Quelle est la teneur en cholestérol des noix de pécan?
KEYWORDS: [[ nutrition, mg, noix ]]

QUERY: Les causes du sous-emploi
KEYWORDS: [[ travailleurs, revenus, pauvreté, croissance ]]

QUERY: Où se trouve danville ca?
KEYWORDS: [[ Californie, Vallée, Comté ]]

QUERY: définition de conundrum
KEYWORDS: [[ énigme, question, difficile ]]
###

<<<
QUERY: user_query
>>>
```

Table 14

Few-shots prompt used for generating keywords related to a provided user query in English.

```
You are a bot used for query expansion.
Your task is to provide a list of 10 keywords, strictly in french, that might be related to a french query
provided after <<<>>>.

You must respond in a single line with the list of 10 keywords in csv format inside [[]].
Do not include the word "KEYWORDS".
Do not provide explanations or notes.
Do not provide translations or other additional information beside the keywords.

###
Here are some examples:

QUERY: which of the following is the main risk factor for cervical cancer?
KEYWORDS: [[ HPV, papillomavirus, immune system, strains ]]

QUERY: how much cholesterol is in pecans
KEYWORDS: [[ nutrition, mg, Nuts ]]

QUERY: causes of underemployment
KEYWORDS: [[ workers, income, poverty, growth ]]

QUERY: where is danville ca
KEYWORDS: [[ California, Valley, County ]]

QUERY: definition for conundrum
KEYWORDS: [[ riddle, question, difficult ]]
###
<<<
QUERY: user_query
>>>
```