

SEUPD@CLEF: Team HIBALL on Incremental Information Retrieval System with RRF and BERT

Notebook for the LongEval Lab at CLEF 2023

Andrea Ceccato¹, Luca Fabbian¹, Bor-Woei Huang¹, Irfan Ullah Khan¹, Harjot Singh¹
and Nicola Ferro¹

¹University of Padua, Italy

Abstract

This report showcases the work of the HIBALL team from the University of Padua on Task 1 LongEval-Retrieval of CLEF 2023 [1] [2]. Our goal was to create a general-purpose information retrieval system, using the CLEF document corpus and judgments as our reference. We explored various approaches, including algorithmic techniques and machine learning methods, and compared their results.

Our best-performing system, a fusion between classical and AI techniques, shows promising outcomes and may serve as a foundation for future developments.

Keywords

CLEF 2023, Information Retrieval, Short-term persistence

1. Introduction

Information retrieval is an open challenge. While working with well-structured data results in precise answers, the information retrieval field often has to handle with documents written by humans, which are usually messy and vague. The state-of-the-art technology is nowhere close to understand the whole meaning of documents, and thus there is great room for improving rank methods.

This is a student group project conducted in the Search Engines course a.y. 2022/23 at the Computer Engineering master degree at University of Padua. In this report, we will compare classical information retrieval system and new AI-powered techniques, and finally draw a new system that combines both. As our ground truth, we used Sub-task A, Short-Term persistence, of the Retrieval task, at LongEval CLEF 2023 Lab [3]. This collection, with a 1570734 document corpus and 672 queries, also includes some judgement and thus is suitable to perform score

CLEF 2023: Conference and Labs of the Evaluation Forum, September 18–21, 2023, Thessaloniki, Greece

✉ andrea.ceccato.6@studenti.unipd.it (A. Ceccato); luca.fabbian.1@studenti.unipd.it (L. Fabbian);


borwoei.huang@studenti.unipd.it (B. Huang); irfanullah.khan@studenti.unipd.it (I. U. Khan);

harjot.singh@studenti.unipd.it (H. Singh); ferro@dei.unipd.it (N. Ferro)

🌐 <http://www.dei.unipd.it/~ferro/> (N. Ferro)

🆔 0000-0001-9219-6239 (N. Ferro)

© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

evaluation.

The paper is organized as follows: Section 2 provides an overview of related works; Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings; finally, Section 6 draws some conclusions and outlooks for future work.

2. Related Work

Information retrieval has been around for a long time, with first experiments being earlier than computers [4]. Despite decades of efforts, the field still has many open challenges. While there exists some ground-solid techniques [5], they fail to capture the true semantic meaning of the documents involved, and thus new research is done every year [6].

Traditional approaches are primary based on text matching, using explicit rules for indexing, querying and ranking algorithms. New approaches rely on AI for semantic extraction. They employ advanced techniques such as Natural Language Processing (NLP).

In this paper, we tried to get the best of both world by combining the two approaches into a hybrid one, and we tested the results on two complete document collections and real queries submitted by users of the Qwant search engine. For query selection and corpora retrieval, we leverage on the work done by CLEF [2].

As a starting point for our code, we used the `frncl/hello-tipster` example provided by Professor Nicola Ferro. The code showcases basic IR features from *parsing documents*, *indexing the parsed documents*, and searching. Everything is performed with Lucene as the underlying engine

For the Re-ranker, we got inspired by JIAZHI GUO's notebook on 'Rescore BM25 with BERT' [7]. For the AI experiments regarding sentence similarity models, we took inspiration from blog tutorials such as [8].

3. Methodology

Our system is an incremental one made of 3 parts, where each part becomes the grounding of the following one:

1. **BASELINE**. First, we performed basic parsing, indexing, and searching using Lucene, a Java library for information retrieval. We tried many runs with different setups.
2. **RRF** (Reciprocal Rank Fusion). Using Python, we merged two standard runs done Lucene Pipeline, one with BM25 Similarity and one with LMDirichletSimilarity. In this way, we got the best recall so far.
3. **BERT**, Re-ranking with Machine Learning. Finally, we took the results obtained previously, and we run second phase re-ranker using Python notebooks.

In parallel, we have done other AI experiments with pre-trained models. The only relevant results, that we have submitted to CLEF, as **AI-FIXED** and **AI-MERGED**, were given by the *distilbert-base-nli-stsb-mean-tokens*, a Sentence Similarity model.

3.1. BASELINE

Our first component, "BASELINE", is where we performed most of the retrieval heavy lifting. This system, written in Java, employs Lucene API under the hood. Here we indexed the entire data-set and produced our first phase ranked list.

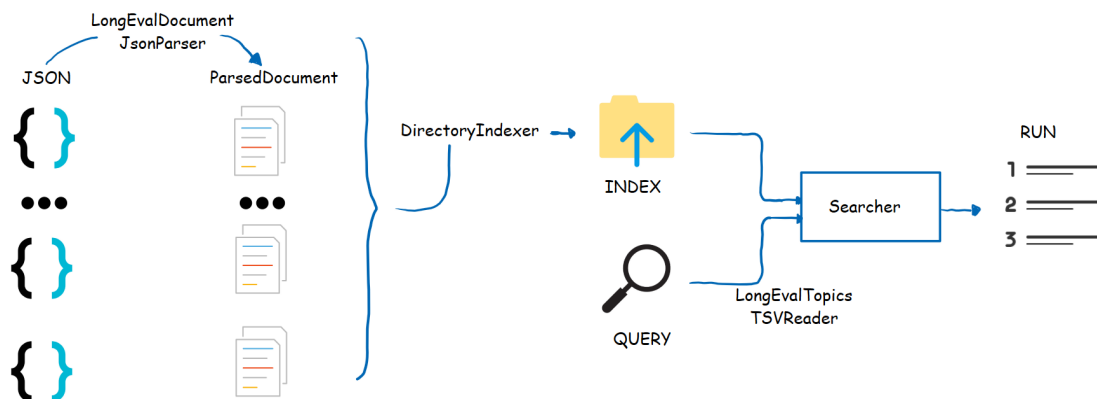


Figure 1: Lucene Pipeline

The system parses the documents thanks to our `LongEvalDocumentJsonParser` and produces `ParsedDocuments` as a result. Then the `DirectoryIndexer` takes those `ParsedDocuments`, produces an Inverted Index, and saves it on the disk. These steps are called Parsing and Indexing. The last step is done by the `Searcher` which loads the Index, parsed the queries, and finally produces the Run. Figure 1 shows these steps.

Parsing: the first step of our Information Retrieval Pipelines is parsing the documents provided by the organizers at CLEF. CLEF corpus is distributed both in the TREC and the JSON formats. We chose the latter, because it is more popular and thus, both Java and Python have libraries to parse it. You may find more details on the data sets in the section 4.1

We wrote our parser, `LongEvalDocumentJsonParser`, which uses JSON Parser provided by "com.fasterxml.jackson.core".

`LongEvalDocumentJsonParser` converts JSON documents in `ParsedDocuments`. These are ready to be used in the subsequent step of the IR chain. We kept the same IDs provided in the JSON document by CLEF.

Meanwhile, we also wrote the `LongEvalTopicsTSVReader` parser for the queries, which were provided in the TSV (tab-separated-values) format.

Indexing: the goal of indexing is to store the document collection in a data structure optimized for lookups. In this way, we will be able to search inside documents as fast as possible. This is done by creating a structure called inverted index, which maps each term to the documents including that term. We used the class `DirectoryIndexer` provided by Professor Ferro as a template, providing as arguments:

- an Analyzer to build `TokenStreams` from a document
- a Similarity function for scoring

Searching: the searcher takes as inputs the indexed documents and the topic, and returns a run file containing the document ranking for each query/topic provided. Analyzer and the Similarity function are also required by the Searcher when creating a run. Searcher and Indexer should have the same Analyzer and Similarity function in a pipeline.

3.1.1. Analyzer:

the analyzer is the core part of every IR pipeline and, as mentioned previously, is required by both the Indexer and the Searcher. Its goal is to build `TokenStreams` from a `ParsedDocument`.

In order to reduce boilerplate code, we created our own Analyzer by extending `org.apache.lucene.analysis.Analyzer`. We added **Tokenizer**, **Stop List**, and **Stemmer** as parameters.

- A **Tokenizer** is the first piece in the pipeline chain. It creates a `TokenStream` from a Reader. We tested `StandardTokenizer`, `WhitespaceTokenizer`, and `LetterTokenizer` and after each tokenizer, we add **LowerCaseFilter** to increase the matched token on search time.
- The **Stop List Filter** aims to get rid of words with low informative value like articles, prepositions, pronouns, and conjunctions. Moreover, this helps to reduce the size of the index (~10GB). We tested Lucene, Lingpipe, Snowball, Okapi, Glasgow, Atire, Terrier, Smart, Zettair, and Indri.
- Another way to increase matching is through Stemming. A **Stemmer Filter** removes prefixes and suffixes, going back to the "root" form of the word. We tested `EnglishMinimal`,

Porter, SnowballPorter, K (Krovetz), and Lovins.

- Other filter we tried are: NGramTokenFilter, ShingleFilter, and LengthFilter.

3.1.2. Extra: Url matching

While looking at the data, we realized some queries are just URLs. This may happen because distracted users mistake the search engine bar for the address bar. In these cases, it's clear the search engine should take the user to the exact website they typed.

This made us think about introducing some special rules just for URL handling. We discovered, however, most of these URLs are not in our dataset: even if we know the website we should point to, we could not, because we do not have a corresponding document ID. Thus, we decided to discard those results. We collected our experiments regarding the topic under `/code/src/main/ai-notebooks/extra-urlmatcher.ipynb`.

Even if we dismissed the experiments, we did it just due to the lack of corresponding data. Perhaps, this could lead to some improvements if investigated in the future.

3.2. RRF (Reciprocal rank fusion)

After testing different combinations of Stemmers and Stop Lists, we focused on **increasing the recall**. In the next phase (see section 3.3) we are performing a re-ranking, and so right now we just care about finding the most relevant documents, and we do not aim at higher precisions.

To increase the recall we tried two Similarities:

- *Okapi BM25 (BM25)* of a document D

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) \frac{f(q_i, D)(k_1 + 1)}{f(q_i, D) + k_1(1 - b + b \frac{|D|}{avgdl})} \quad (1)$$

- *LMDirichletSimilarity* (Bayesian smoothing using Dirichlet priors)

$$p(w_i|d) = \frac{s}{|d| + s} p(w_i|c) + \frac{|d|}{|d| + s} \frac{tf_i(d)}{|d|} \quad (2)$$

and their combinations, as shown in Table 1, with different rank fusion algorithms:

- *Reciprocal Rank Fusion (RRF)* [9]:

$$RRFscore(d \in D) = \sum_{r \in R} 1/(k + r(d)) \quad (3)$$

- *CombSUM*:

$$CombSUM(d \in D) = \sum_{r \in R} r(d) \quad (4)$$

CombSUM is provided by Lucene as the default implementation of MultiSimilarity, but no implementation of RRF was found, so we wrote a Python script, and we used the "fusion" function from *treertools* library for the fusion.

fused_run = fusion.reciprocal_rank_fusion(trec_runs, k, docs_per_query)
 where trec_runs is a list of TrecRun containing the run files.

3.3. BERT

After obtaining as many relevant documents as possible, with the re-ranker our goal was to **increase** the **MAP**, **P@10**, and **nDCG**. See section 4.2 for Evaluation measures.

As our second phase re-ranker we chose *Bidirectional Encoder Representations from Transformers (BERT)* [10], a transformer-based machine learning model.

BERT comes in two model sizes, the small one $BERT_{BASE}$, which has around 110 million parameters, and a large one, $BERT_{LARGE}$ with 340 million parameters [11].

BERT-base-uncased model Given the limited resources, we decided to fine-tune the smaller version of the BERT model for our purpose of re-ranking. The initial model can be downloaded from HuggingFace.

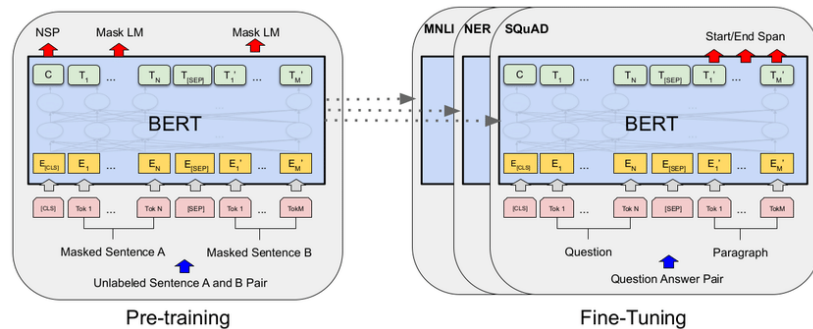


Figure 2: Overall pre-training and fine-tuning procedures for BERT [10]

As input, the model requires a unique vector of tokens (**input_ids**), so for each query we had to concatenate together the query itself and the documents.

The model is able to distinguish the query from the document thanks to **token_type_ids**.

In the following code snippet, we create an input sequence for BERT where we put 0 for query tokens and 1 for document tokens.

```
# make input sequences for BERT
input_ids = query_token_ids + doc_token_ids
token_type_ids = [0 for token_id in query_token_ids]
token_type_ids.extend(1 for token_id in doc_token_ids)
if len(input_ids) > max_input_length: # truncation
input_ids = input_ids[:max_input_length]
token_type_ids = token_type_ids[:max_input_length]
attention_mask = [1 for token_id in input_ids]
```

We can see from the code above that the input tokens, that exceed 512 in length, are truncated to fit the input window size of the BERT-base model.

As a final result, the BERT re-ranker estimates a score on the relevance of a candidate document to a query.

```
score = model(input_ids, attention_mask, token_type_ids)[0]
```

In detail, the steps performed to get this part done are the following ones:

- **BERT Fine-Tuning**

The First part was to fine-tune the BERT-base-uncased model with our training data.

- **Data Pre-processing**

- For each query, we consider training instance a relevant document put together with three other non-relevant documents: one "positive" sample with three "negative" samples.

- The "positive" samples are taken from the CLEF assessments (qrel) file and the "negative" samples are randomly chosen from the set obtained by removing all the relevant documents from the run file created in step 3.2.

- To simplify the model (and ultimately consume less computational power) we did not consider the difference between documents graded as "highly relevant" and "relevant".

We split our training data into training and validation sets with the 80/20 rule.

We used the **Adam optimizer** with an initial learning rate set to $3 * 10^{-5}$. Due to the time constraints, we trained the network with a very low patient of early stopping, set to 2 and the training stopped at the 4th iteration/epoch with the best **loss** on the validation set of **0.58664**.

- **Hyperparameter tuning**

After the model training, we had to tune another **hyperparameter**: *best_bert_weight* for the $BERT_{scores}$ in the ranking. For this we grid searched the *weighted_score* to give to $BERT_{scores}$ and get *best_bert_weight*=**0.012** with the best **MAP** of **0.2298**.

- **Re-ranking**

Finally, after the training and hyperparameter tuning, we could combine the scores from our trained model ($BERT_{scores}$) with the scores from step 3.2 ($RRF60_{scores}$) as per:

```
weighted_scores = RRF60_scores + best_bert_weight * BERT_scores
```

After computing the *weighted_score* of each document, we resort all the documents by the new *weighted_scores* and create the final rank.

3.4. AI Experiments

AI-based technologies are game changers in the current era. Tasks like information retrieval are human related and allow for inaccuracies, so it just feels natural to test AI tools in such projects.

3.4.1. Dismissed experiments

Generative models: The main idea here was to improve or expand document and/or queries thanks to generative text models, such as the one baking ChatGPT. Most of the time, when we search, we are not looking for a document including the question itself, but rather for a document containing the answer to our question. If the AI is able to partially answer a question, or to rephrase a sentence, those new terms could make the query way more precise.

This proved to be a dead-end pretty soon. Weaker models such as gpt2 are totally unable to provide a useful result.

Switching to a more powerful model requires way more hardware resources compared to what we own. The open source competitor of ChatGPT, GPT-j, requires a cluster of GPUs and more than 48 + 16GBs of dedicated RAM.

We then used the expanded query to perform a search with Lucene. The evaluation scored with trec_eval did not show any significant improvements, sometimes it got even worse (query drifting?).

We do believe this approach could lead to great results, however we lack the hardware capabilities to investigate systematically.

Paraphrase models: These models aim to provide a different, improved version of a sentence. They are the AI answer to stemmers (sort of), in a sense that they provide a standardized and less convoluted way to express the same words.

Unfortunately, this process is not deterministic, leading to unexpected behaviours: similar sentences may be mapped into completely different sentences; thus, if we map a document and a query using the same AI model, we may increase their distance instead of reducing it.

Moreover, most of these models (even commercial ones like <https://instatext.io/>), are tuned for long sentences, whereas most of our queries are made of 1–3 words.

No relevant results here as well.

3.4.2. Sentence similarity models

AI-FIXED:

Our biggest experiment employed sentence similarity models. This kind of AI extracts features from text in the form of a dense vector. Similar sentences will have similar corresponding vectors; it means we may perform the actual search using a vector space and pick document vectors closer to the query vector.

After some research, we end up with the *distilbert-base-nli-stsb-mean-tokens* model, which proved to be a good compromise between performance and accuracy. While it is based on BERT as well, it follows a different approach: it is not meant for re-ranking, but rather to provide a ranking on its own.

We were able to run the model and index the whole longeval-train-v2 document collection into a 4.5G file (each document is represented as a 768 float 32 array). This was rather process-intensive, and required hours, even with a 3060 NVidia GPU.

Once indexed, we use another notebook to perform the information retrieval. The full process does not rely on Java/Lucene at all, because it employs Faiss, a powerful python library for working with vector spaces and indexes using hardware acceleration. Faiss is currently the fastest way to look for vector similarity using GPUs, and it's under the hood of popular commercial services such as Amazon Elasticsearch.

Once performed every query, we are finally able to check our results. However, there is a problem: if you look at the results from the `trec_eval` tool, we get very different scores. While the classic approach is rewarded, the AI results are given a **P_10** and even **P_30** of zero.

We think this happens because the AI finds documents different from the one ranked, and thus unknown to `trec_eval`. After all, the train dataset only has like 10 scores per query, and these one have been retrieved using a different approach.

AI-MERGED:

The retrieved documents looked so different from the one suggested by any other method, and this suggested us to try a fusion.

We decided to fuse the "AI-FIXED" run with the most promising one so far, the one described in Chapter 4 as "BASELINE".

Usual fusion functions did not work as expected, so we decided to end up with our own. We chose an almost linear function, because it was easier to tune by hand.

After some attempts, we end up with the following weights:

$$S_{final} = S_{lucene} + \left(\frac{1}{D_{faiss}} - 0.0026 \right) * 1000 * 0.5 * P_{enalty} \quad (5)$$

Where $P_{enalty} = 0.25$ if the document is short, 1 otherwise.

4. Experimental Setup

In this section, we will provide the setup used for each system/run submitted to the CLEF. We will describe the dataset, evaluation measures and the Hardware we used to train and evaluate our system. The 5 submitted runs to the CLEF are:

- **HIBALL_BASELINE**: described in Sec. 3.1 (with Lucene StopList and K Stemmer), is our first and basic system which requires only Lucene to work;
- **HIBALL_RRF60**: described in Sec. 3.2, which merges BM25 and LM1000 similarities with python code;
- **HIBALL_BERT**: described in Sec. 3.3, trained on GPUs with colab notebook;
- **HIBALL_AIFIXED**: described in Sec. 3.4, showcases our sentence similarity model for which we used GPUs from our physical device;
- **HIBALL_AIMERGED**: described in Sec. 3.4, which merges the HIBALL_AIFIXED with the HIBALL_BASELINE.

4.1. Data description

The data provided by CLEF includes a training dataset and a test dataset. The training data can be found at <http://hdl.handle.net/11234/1-5010>, while the test data is at <http://hdl.handle.net/11234/1-5139>.

- **training data**: includes documents, queries and qrels collected during June 2022. The document corpus has 1.5 million scraped Web pages, provided both in the TREC and JSON formats. We worked on the JSON format documents. Each document as an **id** and a **contents** fields. The dataset also provides 672 train queries, in both TREC and TSV formats. In the **qrels** file, documents are labelled with human-provided assessments. Document are scored as either highly relevant (2), relevant (1) or irrelevant (0) to the queries.
- **testing data**: contains two sets of data, short-term dataset collected during July 2022 and long-term dataset collected during September 2022. Queries and documents keeps the same format of the training data.

We used only the English version of the documents for the training and short-term dataset for the evaluation.

4.2. Evaluation Measures

We used the **trec_eval tool** to evaluate the rankings based on query relevance list (**qrels** file), and the rankings of the documents (**run** file). The metrics we focused on are:

- *Average Precision (AP)*: defined as:

$$AP = \frac{1}{RB} \sum_{k \in R} P(k) \quad (6)$$

where RB is the recall base and R is the set of the rank positions of the relevant retrieved documents

- *Mean Average Precision (MAP)*: defined as the mean of the AP scores for each query in a set of Q number of queries:

$$MAP = \frac{\sum_{q=1}^Q AP(q)}{Q} \quad (7)$$

- *Precision at document Cut-off ($P@k$)* with $k = 10$, defined as:

$$P(k) = \frac{1}{k} \sum_{n=1}^k r_n \quad (8)$$

- *Recall at document Cut-off ($R@k$)* with $k=1000$, defined as:

$$R(k) = \frac{1}{RB} \sum_{n=1}^k r_n \quad (9)$$

where RB is the recall base

- *Discounted Cumulated Gain (DCG) $@p$* , defined as:

$$DCG@p = \sum_{i=1}^p \frac{rel_i}{\max(1 + \log_2(i + 1))} \quad (10)$$

- *Normalized Discounted Cumulated Gain ad cut p ($nDCG@p$)*, dividing DCG by ideal DCG ($iDCG$) to normalize the score in $[0, 1]$:

$$nDCG@p = \frac{DCG@p}{iDCG@p} \quad (11)$$

4.3. Repository

For Lucene pipeline, we worked with Java. Additionally, we worked on python notebooks, which have proven to be the best environment for AI prototyping. All the developed code is available on the git repository of the group: <https://bitbucket.org/upd-dei-stud-prj/seupd2223-hiball/src/master/>.

4.4. Hardware

The hardware used to run the experiments are:

- Model: HP
- OS: Windows 11
- CPU: Intel i5
- RAM: 16GB
- Storage: SSD 512GB

-
- Model: MacBook Air
 - OS: macOS Ventura 13.3.1
 - CPU: M1
 - RAM: 16GB
 - Storage: SSD 256GB

A GPU is mandatory, since AI processes rely on hardware acceleration to run in feasible time. For the RE-RANKER, training and getting the re-ranked runs, we used free tier of colab which includes:

- Model: Google Compute Engine
- RAM: 12.7 GB
- GPU: NVIDIA® T4 15.0 GB
- Storage: SSD 78.2 GB

The Google Colab free tier is not enough to run the ai experiments of section 3.4, but fortunately we had the opportunity to run our codes on a physical device:

- Model: Dell
- RAM: 16 GB
- GPU: NVIDIA® 3060
- Storage: SSD 1024 GB

5. Results and Discussion

In this section, we will first introduce some general considerations about each employed technique. We will then provide a result comparison (5.5) based on trec_evals metrics, and finally provide a statistical analysis of it (5.6).

5.1. BASELINE System

To choose our BASELINE system, we experimented all the possible combinations of stemming and StopList filters with BM25 Similarity.

The results were not significantly different among them.

- The **map** values were around **0.1465**;
- the **P@10** values were around **0.0937**;
- **Recalls** were around **0.6954**;
- and **nDCG** values are around **0.2896**.

K-stemming is slightly outperforming Lovins-stemming and Porter-stemming. While "Lucene" and "Lingpipe" Stop Lists have the best overall performances among all stop lists we tested in combination with the stemmers. For the BASELINE system, we chose Lucene StopList and K Stemmer. With this configuration, we have the highest MAP and P@10 values.

5.2. RRF (Reciprocal Rank Fusion)

In the second part of our system development, where the goal was to improve recall, we tried out different Similarities and their combinations. From the Table 1, we can see the best recall is achieved with RRF rank fusion algorithm, and we chose to use RRF60 for the re-ranking system as it has higher overall performance map value.

Table 1

The results with different Similarities and their combination on the train data

runs	MAP	P@10	recall@1000	nDCG
luceneStop-kStem-bm25 (BASELINE)	0.1498	0.0952	0.705	0.2939
luceneStop-kStem-LmDir1000	0.13	0.0838	0.6863	0.2723
luceneStop-kStem-LmDir2000	0.1232	0.08	0.6697	0.2625
luceneStop-kStem-bm25Lm1000-comboSum	0.1387	0.0888	0.7042	0.2834
luceneStop-kStem-bm25Lm1000-RRF0	0.1423	0.0909	0.7135	0.2891
luceneStop-kStem-bm25Lm1000-RRF60	0.1429	0.0905	0.7135	0.2889
luceneStop-kStem-bm25Lm1000-RRF80	0.1428	0.0902	0.7135	0.2888
luceneStop-kStem-bm25Lm1000-RRF100	0.1427	0.09	0.7135	0.2887

5.3. BERT

From the results in Table 2, we can see that with our HIBALL-BASELINE, with Lucene StopList-Filter and KStemFilter, we improved slightly the baseline of LongEval-Retrieval [3]. However, from Tables 2, 3, and 4 we can see that the re-ranking system with BERT model, which is based on RRF60 system, significantly improves all the measures MAP, P@10 and nDCG and Recall.

The BERT model fine-tuned with the parameters described in Sec. 3.3 and the implementation of early stopping show a good generalization of the model on new data. For example, the results on TEST data (unknown to the model) in Tab. 4, are similar to those on the TRAIN data (used for the training and validation) in Tab. 2.

5.4. AI experiments

We had trouble measuring the final score. Evaluating our results has been and still is a challenge because the train set provided by CLEF is clueless about some documents retrieved by AI models, while the same documents look right at a human eyes. Sometimes, the conclusion we draw by manually looking at the results were clearly conflicting with the one measured by trec_eval.

We also realized too late the HIBALL_BERT was so successful, because it did not look so promising at first. Since we fused with HIBALL_BASELINE, the improvements we got are related to the HIBALL_BASELINE and do not look so good compared to the HIBALL_BERT ones.

5.5. Result comparison

Tab. 2, Tab. 3, Tab. 4 show the results of each submitted run on TRAIN, HELDOUT and TEST data respectively. By comparing the tables, we can see that HIBALL_BERT is clearly the one leading on every dataset, followed by the AI merged with BASELINE (HIBALL_AIMERGED). Then our BASELINE system, followed by HIBALL_RRF60. Lastly, we have the AI system (HIBALL_AIFIXED), which has very poor performance independently. The tables are ordered by decreasing MAP values, but we can see that HIBALL_RRF60 has the same Recall values of the best performing system, as expected.

Table 2

The results with different configuration on the TRAIN DATA

runs	MAP	P@10	nDCG	Recall
HIBALL_BERT	0.2298	0.1429	0.3747	0.7135
HIBALL_AIMERGED	0.1501	0.0927	0.2943	0.7047
HIBALL_BASELINE	0.1498	0.0952	0.2939	0.705
HIBALL_RRF60	0.1429	0.0905	0.2889	0.7135
HIBALL_AIFIXED	0.0262	0.0210	0.0891	0.2975

Table 3

The results with different configuration on the HELDOUT DATA

runs	MAP	P@10	nDCG	Recall
HIBALL_BERT	0.1732	0.1163	0.3119	0.6500
HIBALL_AIMERGED	0.1363	0.0888	0.2790	0.6402
HIBALL_BASELINE	0.1255	0.0888	0.2652	0.6402
HIBALL_RRF60	0.1247	0.0878	0.2664	0.6500
HIBALL_AIFIXED	0.0332	0.0204	0.0908	0.2542

Table 4

The results with different configuration on the TEST DATA

runs	MAP	P@10	nDCG	Recall
HIBALL_BERT	0.2000	0.1200	0.3433	0.6939
HIBALL_AIMERGED	0.1576	0.0954	0.2955	0.6891
HIBALL_BASELINE	0.1512	0.0958	0.2901	0.6913
HIBALL_RRF60	0.1469	0.0920	0.2879	0.6939
HIBALL_AIFIXED	0.0320	0.0210	0.0924	0.2927

5.6. Statistical Analysis

While table 4 shows the means of the measures, the boxplots depict better the performance distribution over the whole data. Figure 3 shows the results on the HELDOUT set, while figure 4 reveals the results on the test set. From the boxplot of the measure of AP (figure 3a), we noticed that HIBALL_BERT with more high-precision queries outperforming other systems, despite the fact that there are still some queries' precision close to 0. The distribution of the nDCG boxplot (figure 3b) shows a similar trend. That tells us that the BERT re-ranker improved the AP and nDCG of many queries, but there are still many queries' results do not benefit from the BERT re-ranker.

Unsurprisingly, the boxplots of the results on the test set are in like manner with that on the heldout set. In figure 4a, we can see that HIBALL_BERT system has a much larger third quartile toward higher AP value compared with BASELINE. The AP of HIBALL_BERT system has a certain ratio of queries with high AP values above 0.6, which significantly raises the overall MAP performance. The other 3 runs, HIBALL_BASELINE, HIBALL_RRF60 and HIBALL_AIMERGED, do not show much difference. A similar behavior can be observed in the nDCG boxplots (figure 4b). It is worth noticing that, from the heldout set to the test set, HIBALL_BERT's third quartile of the P@10 distribution stays at around 0.2 (figure 4c), whereas the BASELINE model's third quartile of P@10 drops from 0.2 to 0.1. Thus, the IR-system with BERT re-ranker sustains its high performance of the P@10 on the TEST set.

The ANOVA tests were implemented to compare the 5 runs. The p-values of the two-way ANOVA tests of AP and nDCG on the HELDOUT set are close to 0 (table 5 and 6), while the p-values on the TEST set are exactly 0 as shown in table 7 and 8. The close to 0 p-values are far below the threshold $\alpha = 0.05$. We confidently reject the null hypothesis, i.e., the 5 runs are equal, and claim that the runs are different regarding the measure AP and nDCG.

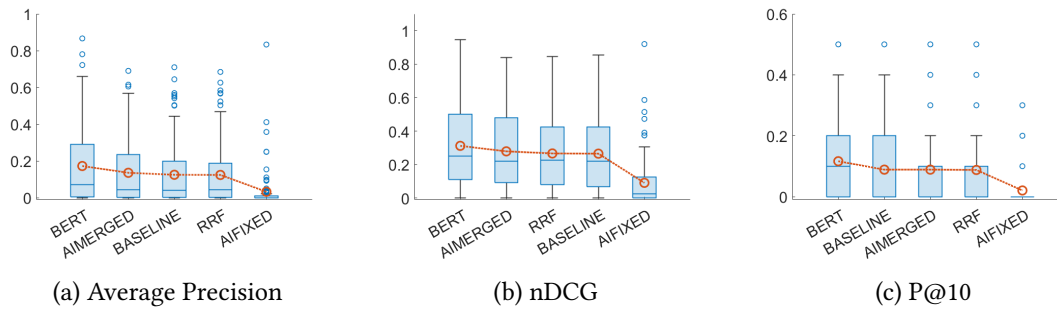


Figure 3: Boxplots of the 5 runs in the decreasing order of the mean performance on the HELDOUT set.

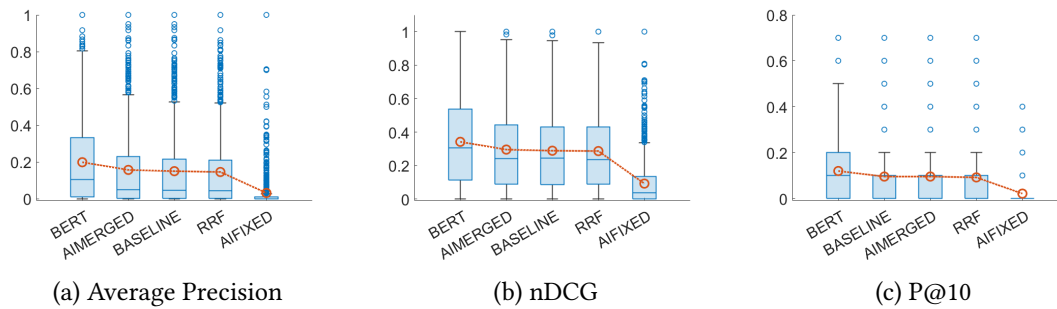


Figure 4: Boxplots of the 5 runs in the decreasing order of the mean performance on the TEST set.

In figure 5, the Tukey’s *Honestly Significant Difference (HSD)* clearly reveals the difference of the means among the 5 runs on HELDOUT set. HIBALL_BERT has high AP and nDCG means, while HIBALL_BASELINE, HIBALL_RRF60 and HIBALL_AIMERGED have similar means and variance, which reinforce the argument inferred from the boxplots shown in figure 3. Once again, the Tukey’s HSD in figure 6 shows the difference of the means among the 5 runs on the TEST set. Therefore, we can safely conclude that the 5 runs are significantly different.

Table 5

Two-way ANOVA analysis of the 5 runs on AP of the HELDOUT set

Source	SS	df	MS	F	Prob>F
Runs	1.0455	4	0.26138	23.92	1.06341e-17
Measures	10.337	97	0.10657	9.75	5.30974e-61
Error	4.2404	388	0.01093		
Total	15.623	489			

Table 6

Two-way ANOVA analysis of the 5 runs on nDCG of the HELDOUT set

Source	SS	df	MS	F	Prob>F
Runs	2.9632	4	0.74081	52.87	1.53768e-35
Measures	18.6551	97	0.19232	13.73	1.36549e-80
Error	5.4367	388	0.01401		
Total	27.055	489			

Table 7

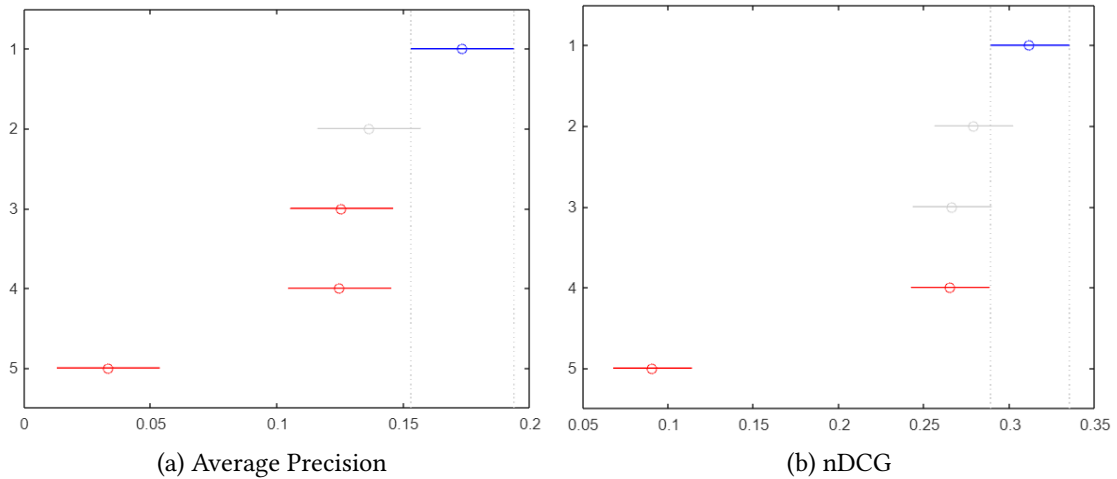
Two-way ANOVA analysis of the 5 runs on AP of the TEST set

Source	SS	df	MS	F	Prob>F
Runs	13.75	4	3.43756	228.55	0
Measures	121.512	881	0.13792	9.17	0
Error	53.002	3524	0.01504		
Total	188.265	4409			

Table 8

Two-way ANOVA analysis of the 5 runs on nDCG of the TEST set

Source	SS	df	MS	F	Prob>F
Runs	33.172	4	8.29291	507.17	0
Measures	183.696	881	0.20851	12.75	0
Error	57.622	3524	0.01635		
Total	274.49	4409			

**Figure 5:** Tukey's HSD test of the 5 runs on the HELDOUT set

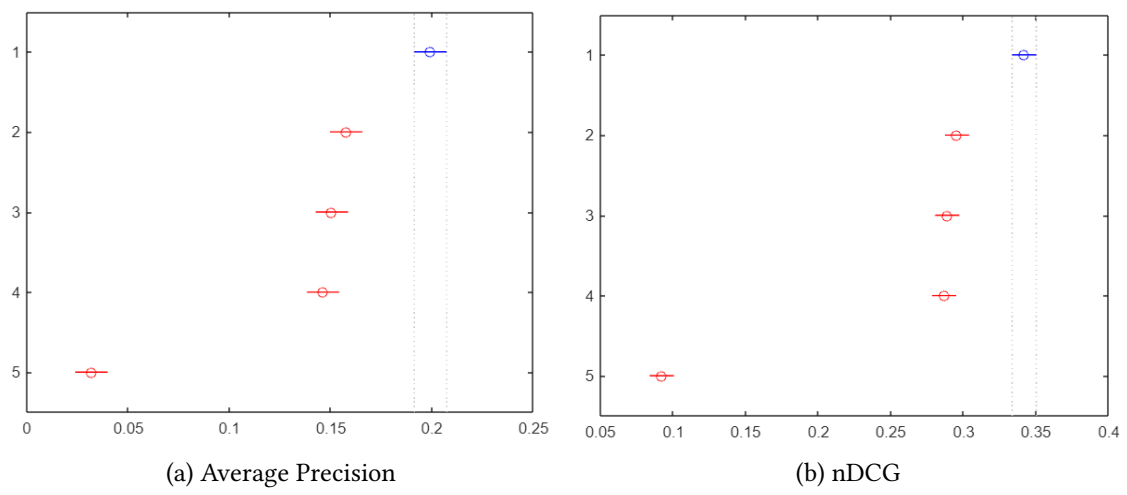


Figure 6: Tukey's HSD test of the 5 runs on the TEST set

6. Conclusions and Future Work

In this report, we started by introducing a relevant problem in the Information Retrieval domain, and we proposed our System as a solution. Then we explained how we incrementally came up with the final System, i.e., starting from a BASELINE system and then trying to improve recall with RRF, and finally, other measures like MAP, P@10, and nDCG with BERT.

Finally, we spent some time investigating AI models pretrained on general purpose text. Those provided different results from the classical approach; while those results looks promising on manual inspection, they were outside the assessments' radar, and thus we were not able to provide a solid measure for them.

In conclusion, from our experiments, BERT re-ranking is the most promising way to improve the search engine metrics. There are, however, flaws in our BERT-base training processes. Firstly, due to the limitation of the GPU and RAM memory, we only feed first 512 tokens to the BERT-base model, which means the model was not trained with sentences large enough for each document (4900 on average). Secondly, the training process is time-consuming, restricted by not enough powerful GPU. We trained the BERT-base model for four epochs only, and we did not make distinction between "highly relevant" and "relevant" documents assessments.

We could also train the BERT-base model with full training documents by shifting the input window. Moreover, the BERT-base mode can be trained with more epochs to get higher accuracy. To further improve the BERT re-ranker performance, the BERT-large model might be a better option than the BERT-base model.

We should also find a way to measure if the pretrained AI experiments were actually on the spot or not, and try to fuse it with the HIBALL_BERT instead of the HIBALL_BASELINE run.

References

- [1] R. Alkhalifa, I. Bilal, H. Borkakoty, J. Camacho-Collados, R. Deveaud, A. El-Ebshihy, L. Espinosa-Anke, G. Gonzalez-Saez, P. Galuščáková, L. Goeuriot, E. Kochkina, M. Liakata1, D. Loureiro, H. T. Madabushi, P. Mulhem, F. Piroi, M. Popel, C. Servan, A. Zubiaga, Overview of the clef-2023 longeval lab on longitudinal evaluation of model performance, in: *Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Fourteenth International Conference of the CLEF Association (CLEF 2023)*, Lecture Notes in Computer Science (LNCS), Springer, Thessaliniki, Greece, 2023.
- [2] Clef 2023, 2023. URL: <https://clef-longeval.github.io/tasks/>.
- [3] P. G. R. Deveaud, G. Gonzalez-Saez, P. Mulhem, L. Goeuriot, F. Piroi, M. Popel, Longeval-retrieval: French-english dynamic test collection for continuous web search evaluation, 2023. *arXiv:2303.03229*.
- [4] V. Bush, et al., As we may think, *The atlantic monthly* 176 (1945) 101–108.
- [5] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, A. Rummler, An exploratory study of information retrieval techniques in domain analysis, in: *2008 12th International Software Product Line Conference, IEEE, 2008*, pp. 67–76.

-
- [6] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, M. Brockschmidt, Codesearchnet challenge: Evaluating the state of semantic code search, arXiv preprint arXiv:1909.09436 (2019).
 - [7] Rescore bm25 with bert, 2020. URL: <https://www.kaggle.com/code/jerrykuo7727/rescore-bm25-with-bert/notebook>.
 - [8] K. Stathoulopoulos, How to build a semantic search engine with transformers and faiss, 2021. URL: <https://towardsdatascience.com/how-to-build-a-semantic-search-engine-with-transformers-and-faiss-dcbea307a0e8>.
 - [9] G. V. Cormack, C. L. A. Clarke, S. Buettcher, Reciprocal rank fusion outperforms condorcet and individual rank learning methods, in: SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, 2009, pp. 758–759. URL: <http://doi.acm.org/10.1145/1571941.1572114>.
 - [10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. arXiv:1810.04805.
 - [11] Bert (language model), 2023. URL: [https://en.wikipedia.org/wiki/BERT_\(language_model\)](https://en.wikipedia.org/wiki/BERT_(language_model)).