A Comparative Study of Lexical, Neural, and Hybrid Retrieval Strategies for the LongEval-SciRetrieval Task

Adrian Bergler[1,2], Margarida Maria Agostinho Gonçalves[1,3], Christine Hubinger[1,4], Dmytro Pashchenko[1,5], and Max Tiessler[1,6]

[1] TU Wien, Austria
[2] e1646819@student.tuwien.ac.at
[3] e12142151@student.tuwien.ac.at
[4] e9526158@student.tuwien.ac.at
[5] e12307842@student.tuwien.ac.at
[6] e12234573@student.tuwien.ac.at

Abstract. The LongEval-SciRetrieval task, a component of the LongEval 2025 Lab, part of the CLEF 2025, provides a platform for evaluating Information Retrieval (IR) systems within the domain of scientific literature, focusing on temporal robustness while using the CORE open-access publications dataset. This report presents a detailed description of our group's systematic exploration and comparison of three different IR methodologies. Our initial project design, a complex but flexible software pipeline, evolved into standalone executable components due to the complexity of large-scale data processing and the mix of team technical expertise. We began by establishing a baseline with a BM25 model, exploring its theoretical foundations, designing a text preprocessing pipeline, and conducting hyperparameter optimization to improve performance. Afterwards, we explored neural representation learning for end-to-end semantic retrieval. First, this phase was implemented using fine-tuning of SciBERT, a language model pre-trained on scientific text. Second, motivated by its state-of-the-art performance on benchmarks such as the MS MARCO leaderboard, we implemented a second version using ColBERTv2, employing the RAGatouille library for document indexing and search. Finally, we implemented our most advanced technique using a two-stage hybrid re-ranking architecture. This technique relied first on BM25 for efficient initial candidate generation, followed by semantic re-scoring of these candidates by a pre-trained ColBERTv2 model. This report discusses our methodology, the evolution of our implementation strategy, the theoretical background that guided our model choices, and the practical adaptations that had to be conducted due to significant computational and infrastructure constraints. We finally present performance analyses on development data and detail our submissions to the TIRA evaluation platform. The complete codebase is available via GitHub.

Keywords: Information Retrieval · Scientific Document Retrieval · BM25 · Neural Ranking · Semantic Search · SciBERT · ColBERT · Re-ranking · LongEval · Click Models · Evaluation Metrics · RAGatouille.

1 Introduction

The exponential growth of scientific literature [1] presents both an opportunity and a significant challenge for researchers seeking to stay abreast of advancements and discover relevant prior work. Therefore, effective Information Retrieval (IR) tools are essential in this context to ensure knowledge discovery and accelerate scientific progress. The LongEval-SciRetrieval task, as the second task within the LongEval 2025 Lab [2], is specifically designed to help research and evaluate IR systems in this domain. It is backed by a collection of open-access scholarly documents from the CORE (Connecting Repositories) aggregation. A primary objective of the LongEval initiative, as stated on its official website and in its overview paper [2,3], is to assess the temporal robustness of IR systems (i.e., how their performance is maintained or degrades as the underlying document collection changes over time).

Relevance assessments for the LongEval-SciRetrieval task are derived from models of user click behavior, as detailed by the task organizers. This approach, more common in web search evaluation [4], uses implicit user feedback for explicit relevance judgments. The provided data includes search information (anonymized user sessions, queries, and ranked lists of results) and click information (clicked

links and their positions). However, since the scientific papers iteration of the task is newly organized and thus has fewer dataset snapshots compared to other LongEval tracks, the long-term vision involves evaluating systems across multiple time-stamped versions of the collection.

The evaluation is based on two primary aspects, as specified in the task guidelines:

1. **nDCG (Normalized Discounted Cumulative Gain) scores:** Calculated on the provided test sets, this metric evaluates the quality of the ranked list, with a discount factor that emphasizes the importance of retrieving relevant documents at higher ranks.
2. **Relative nDCG Drop (RnD):** This metric measures the difference in nDCG scores between system runs on different temporal snapshots of the test collection, thereby quantifying the impact of data evolution on retrieval performance.

Although the training data for our current task was limited to a single time point, limiting an analysis of RnD, our efforts focused on developing and comparing retrieval systems that can serve as a foundation for future longitudinal studies within this framework.

The dataset, sourced from CORE, was made available via the TU Wien Research Data repository [5]. The training corpus comprised 2,014,265 scientific articles (available as abstracts or full-text, with the latter amounting to approximately 60 GB), 393 queries, and 4,262 click-derived relevance judgments [5]. The test corpus consisted of 1,213,728 articles, evaluated against two query sets: one from November 2024 (99 queries) and another from January 2025 (492 queries) [6].

This paper details our systematic exploration of three distinct IR paradigms:

1. A traditional lexical model, BM25, serving as a strong and well-understood baseline (Section 3).
2. Neural representation learning models for end-to-end semantic retrieval, specifically investigating a fine-tuned SciBERT model and a pre-trained ColBERTv2 model integrated via RAGatouille (Section 4).
3. A hybrid two-stage re-ranking architecture, aiming to combine the strengths of lexical recall from BM25 with the semantic precision of ColBERTv2 (Section 5).

We detail our methodology, including its evolution from an initial comprehensive pipeline design to more agile, task-focused implementations (Section 2). We further discuss the significant computational challenges encountered (Section 2.1), present our performance on development data alongside a discussion of the results (Section 6), detail our submissions to the TIRA platform (Section 8), and outline our contributions (Section 7). All code developed for this work is accessible via a public GitHub repository: https://github.com/mtiessler/188.980_Advanced_Information_Retrieval_Project.

2 Methodology Evolution and General Setup

Our initial project idea was to implement a highly modular, generic pipeline focused on reusability and clear separation of concerns, using distinct modules for data loading, text preprocessing, various retrieval algorithms, and standardized evaluation. The components of this pipeline were developed primarily for the BM25 baseline. The idea behind this initial approach was to create a robust and extensible framework that could integrate various IR experiments in a flexible way and without effort.

However, as we integrated more complex neural models and dealt with the large volume of the LongEval-SciRetrieval dataset plus our limited infrastructure, made this approach impractical. The overhead of maintaining and extending the generic pipeline for experimental needs, specially the integration of multiple libraries and managing their specific dependencies, added to considerable skill gap in software engineering expertise within the team, led us to make a change. We pivoted to more focused, isolated Jupyter Notebooks for the neural network experiments and the re-ranking pipeline, so they could be easily manipulated and exported between environments. This also allowed faster prototyping and iteration for these specific tasks.

The old pipeline can be found in the `\deprecated` folder within the repository. The final task modules can be found in the `\pipeline` directory.

2.1 Computational Environment and Resources

We started using our own hardware, but once we saw that it was not sufficient for this type of assignment, we began relying on the TU Wien JupyterHub cluster, which was adequate for initial BM25 development. However, a blocking issue occurred while trying to use GPU acceleration for neural models: the cluster's environment lacked a readily available NVIDIA CUDA Compiler (`nvcc`). This compiler was needed for building custom CUDA C++ extensions, which are necessary for the performance of libraries like `faiss-gpu` (used in the background by RAGatouille for efficient dense vector indexing) and for compiling ColBERT's own optimized execution kernels from source. Without the CUDA compiler, these libraries failed and fell back to CPU implementations, rendering large-scale neural experiments impractical in that environment.

To overcome this, we migrated our neural model experimentation to Google Colab Pro (paid service), which provided access to high-RAM environments with GPUs like the NVIDIA T4 and, for some larger ColBERT tasks, A100 GPUs equipped with the full CUDA toolkit. This environment change enabled us to use GPU acceleration. However, it also introduced new limitations: Google Colab Pro has session runtime limits that, with the full-text dataset, caused execution to stop due to long runtimes, and it also incurs monetary costs for sustained high-end GPU usage (we estimated more than EUR 2 per ColBERT execution attempt on an A100 instance). These resource limitations (both time and cost) influenced our experimental design, forcing us to focus ColBERT experiments on the abstract set rather than the significantly larger full-text collection, resulting in lower performance.

2.2 Key Libraries and Software Tools

Our implementation used a combination of Python libraries for IR and NLP; the following list highlights the most important ones:

- **NLTK (Natural Language Toolkit) & PyStemmer:** These were mainly used for preprocessing. NLTK was used for tasks like sentence splitting, word tokenization (`word_tokenize`), and providing standard English stopword lists. `PyStemmer` provided the Snowball stemmer.
- **rank_bm25:** This library offered an efficient and straightforward Python implementation of the BM25 ranking function, allowing us to quickly establish a strong lexical baseline.
- **Hugging Face transformers:** It provided access to datasets and pre-trained models, including SciBERT and ColBERT, along with their associated tokenizers that were needed for converting text into model-understandable input.
- **RAGatouille (v0.0.9):** To simplify the deployment of the complex ColBERT model, we used RAGatouille [7]. This library serves as a wrapper around ColBERT, abstracting the complexity involved in indexing document collections and performing ColBERT's characteristic late-interaction search.
- **ir_measures:** This package was used to compute the different IR evaluation metrics, so that our results were comparable and correctly calculated.

3 Approach 1: Lexical Retrieval with BM25 (Baseline)

BM25 ranks a set of documents based on the query terms appearing in each document, without considering the relationships between the terms (e.g., word order). It is a probabilistic model that builds upon the ideas of term frequency (TF) and inverse document frequency (IDF).

3.1 Preprocessing for BM25

Effective preprocessing is important in lexical models like BM25 to normalize text and reduce the vocabulary size to its most informative terms. Our pipeline included:

1. **Text Extraction:** For each document, the title, abstract, and—if available—the full-text were concatenated into a single textual representation.
2. **Text Normalization:** Conversion to lowercase ensures that terms are treated case-insensitively. Punctuation and single digit removal simplify tokens and reduce noise.

3. **Tokenization:** NLTK's `word_tokenize` was used to split text into individual words. During development, alternative tokenizers such as those from `Pyserini` and `spaCy` were evaluated but ultimately rejected, as they did not lead to measurable improvements in retrieval performance.
4. **Stopword Removal:** Common, high-frequency English stopwords were removed using NLTK's list, as these are not semantically important.
5. **Stemming:** The Snowball stemmer (`PyStemmer`) was applied to reduce words to their morphological root (e.g., "retrieval", "retrieving" → "retriev").

The large document collection was indeed a challenge for efficient preprocessing, especially without relying on big data and distributed tools such as Apache Spark or Apache Kafka (which was definitely out of scope for this subject and report).

Nevertheless, our BM25 implementation contained chunk-based tokenization and cached tokenized document representations to disk. Figure 1 gives an insight on the resource usage during the BM25 pipeline (implemented in the `pipeline/task_1` directory).

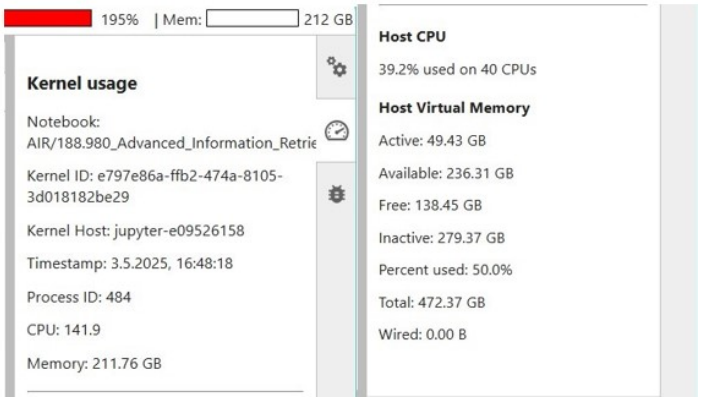

Fig. 1. Snapshot of resource usage BM25 preprocessing and ranking

3.2 BM25 Model: Theory and Implementation

The BM25 scoring function for a document D and a query $Q = \{q_1, \ldots, q_n\}$ is:

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

where $\text{IDF}(q_i)$ is the inverse document frequency, $f(q_i, D)$ is the term frequency, $|D|$ is the document length, and avgdl is the average document length. k_1 controls term frequency saturation, and b controls length normalization. We used the `BM25Okapi` implementation from the `rank_bm25` package.

Another notable implementation of BM25 is BM25-Sparse [12] `BM25S` which offers a fast implementation of BM25 scoring. However, due to technical difficulties in achieving reliable execution within Jupyter and Google Colab environments, we ultimately decided against integrating this library into our experiments [13]. Nevertheless, it remains a promising project under active development and may be worth exploring in future work.

3.3 Hyperparameter Tuning for BM25

The parameters k_1 and b are corpus-dependent. We made a grid manual search for finding the optimal nDCG@10 on development data (Figure 2 highlights visually the different performances we got for each combination):

- **Abstracts**: $k_1 = 1.0$, $b = 0.7$ ⇒ nDCG@10 = 0.1778
- **Full-text**: $k_1 = 0.2$, $b = 0.8$ ⇒ nDCG@10 = 0.2002

Table 1. BM25 Hyperparameter Tuning (nDCG@10) on Abstract Development Set (Selected Values).

nDCG@10	**b = 0.65**	**b = 0.70**	**b = 0.75**
k = 0.95	0.1765	0.1775	0.1771
k = 1.00	0.1764	**0.1778**	0.1772
k = 1.05	0.1760	0.1776	0.1769

Table 2. BM25 Hyperparameter Tuning (nDCG@10) on Full-text Development Set (Selected Values).

nDCG@10	**b = 0.70**	**b = 0.75**	**b = 0.80**	**b = 0.85**
k = 0.10	0.1972	0.1998	0.1996	0.1985
k = 0.20	0.1973	0.2000	**0.2002**	0.1988
k = 0.30	0.1971	0.1998	0.1999	

These results indicate that optimal BM25 parameters vary by document type. For abstracts, higher k_1 suggests that term frequency remains informative, likely due to the concise nature of abstracts. Alternatively, the lower k_1 for full-text implies quicker saturation of term frequency, which is expected in longer documents where repetition is more common. Slightly higher b values in full-text also suggest stronger normalization by document length.

For a standalone BM25 submission we used $k_1 = 0.2$, $b = 0.8$ as the most promising candidate.

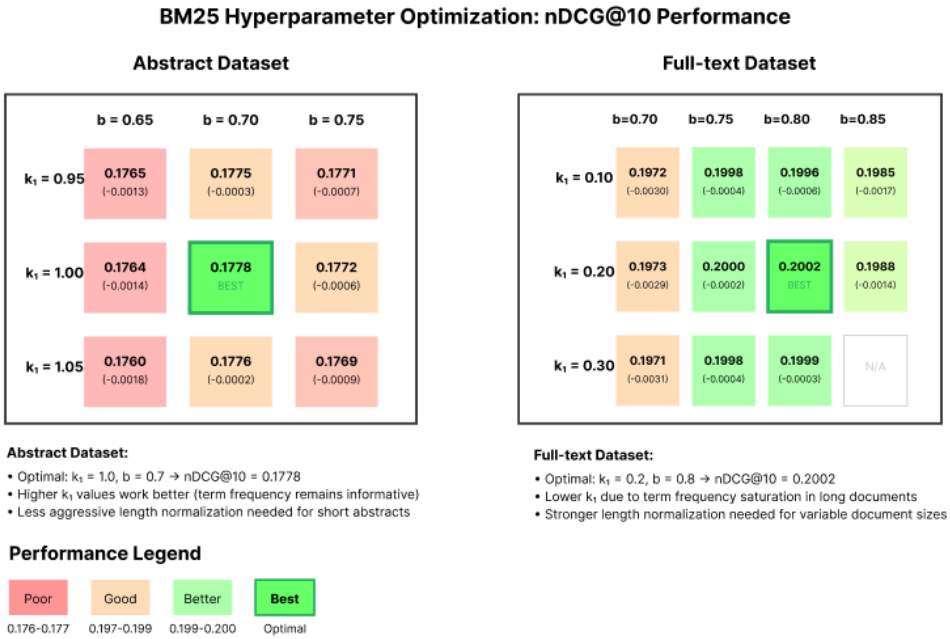

Fig. 2. BM25 hyperparameter optimization results showing nDCG@10 performance across different k_1 and b values.

3.4 BM25F

Due to the documents in the corpus containing title, abstract and full-text content, that might have a different impact on the relevance of a document in the context of a query, it was decided to also attempt some experimentation using BM25F, since it provides parameters to weigh different streams of the document data.

The BM25F scoring function for a document d and a query q is:

$$\text{BM25F}(q,d) = \sum_{t \in T_d \cap T_q} \frac{\widetilde{tf}_{t,d}}{k_1 + \widetilde{tf}_{t,d}} \cdot \log \frac{|D| - df_t + 0.5}{df_t + 0.5}$$

$$\widetilde{tf}_{t,d} = \sum_{s=1}^{S_d} w_s \cdot \frac{tf_{t,s}}{(1-b_s) + b_s \dfrac{sl_s}{\text{avgsl}}}$$

where $\widetilde{tf}_{t,d}$ is the field-weighted term frequency of term t in document d, $tf_{t,s}$ is the raw term frequency of t in stream s, sl_s is the length of stream s, avgsl is the average stream length in the index, w_s is the weight assigned to stream s, b_s controls length-normalization for stream s, k_1 controls term-frequency saturation, $|D|$ is the total number of documents, and df_t is the number of documents in which term t appears.

We initially selected PyTerrier for our experiments due to its reported efficiency and speed. Although we conducted preliminary tests with the framework, we ultimately did not adopt it in our final pipeline. This decision was primarily due to compatibility issues with the Java environment on the TUCluster, since PyTerrier requires access to a fitting JDK. The corresponding prototype implementation is available in the `deprecated/bm25f_prototype` directory.

Also the limited structure of the dataset raises questions about the usefulness of applying BM25F. Our approach makes use of only the title, abstract, and full-text fields, with the full-text already cleaned and lacking additional structural annotations. Simply separating these fields may not lead to notable improvements in retrieval performance, and we expect that the potential benefits of field-aware ranking are higher if there are further fields.

4 Approach 2: Neural End-to-End Retrieval

Neural Information Retrieval (IR) models learn dense vector representations that capture semantic meaning of queries and documents. This addresses vocabulary mismatch, where relevant documents lack exact keywords from the query but remain semantically related. The fundamental shift from lexical to neural retrieval represents a move from exact term matching to semantic similarity in high-dimensional vector spaces, enabling retrieval of conceptually related documents even when surface-level text differs significantly.

We explored neural end-to-end retrieval through two models: a general-domain cross-encoder, followed by domain-specific approaches using SciBERT and ColBERTv2.

4.1 Initial Dense Retrieval (MS Marco Cross-Encoder)

We tested the `cross-encoder/ms-marco-MiniLM-L-6-v2` model as an exploratory trial. This cross-encoder processes query and document text jointly to produce relevance scores, contrasting with bi-encoder approaches that encode queries and documents separately. Cross-encoders typically achieve higher accuracy due to their ability to model query-document interactions but are computationally expensive for large-scale retrieval.

While effective on general datasets like MS MARCO, it achieved an nDCG@10 score of 0.0234 on our scientific literature dataset. This extremely low score indicates the model fails to identify relevant documents in the top 10 results, demonstrating severe domain mismatch and confirming the need for models trained on scientific text. The poor performance highlights a critical challenge in neural IR: models trained on web search data often fail to transfer to specialized domains without adaptation. This implementation can be found integrated in `deprecated/pipeline.py`.

4.2 SciBERT for Relevance Classification

We selected SciBERT (`allenai/scibert_scivocab_uncased`) [8] to address domain specificity. SciBERT is pre-trained on 1.14M papers from Computer Science and biomedical domains, making it suited to scientific language and concepts. The model uses a scientific vocabulary (SciVocab) derived from scientific text, which better represents domain-specific terminology compared to general vocabulary. Fine-tuning SciBERT as a sequence classifier for relevance prediction leverages its scientific domain knowledge for accurate query-document relevance determination. The model predicted a single continuous relevance score (`num_labels=1`).

Implementation and Training Input consisted of query text concatenated with document abstract, separated by `[SEP]` tokens (e.g., "query text [SEP] document abstract text"). This concatenation approach allows the model to learn query-document interactions through attention mechanisms. Moreover, combined sequences were tokenized using the SciBERT tokenizer, with padding and truncation to 512 tokens, which is the token length for BERT models.

Training relevance judgments (qrels) were split into training (90%) and evaluation (10%) sets using stratified sampling by query IDs to ensure representative distribution. The model was fine-tuned for 5 epochs using AdamW optimizer with learning rate 3×10^{-5} and batch size 128. Mixed-precision training was also added to accelerate computation and reduce memory usage. The implementation of this experiment is available in `pipeline/task_2/BERT_v1.ipynb`.

SciBERT Performance Table 3 shows the fine-tuned SciBERT performance on the abstract development set. The nDCG@10 score of 0.1896 represents a substantial improvement over the general-domain model but indicates that only about 19% of the ideal ranking quality is achieved. The low MAP score (0.0999) suggests poor precision across all relevant documents, while the higher MRR (0.7564) shows that when relevant documents are found, they typically appear in the top few results. These mixed results represent SciBERT's domain knowledge benefits but highlight too the challenge of the lab task (i.e., scientific document retrieval).

Table 3. SciBERT Fine-tuning Performance on Abstract Dataset.

Metric	Score
nDCG@10	0.1896
AP (MAP)	0.0999
RR (MRR)	0.7564

4.3 ColBERTv2 with RAGatouille for End-to-End Retrieval

As we were motivated with our outcomes, we explored ColBERTv2 (`colbert-ir/colbertv2.0`) [9] after observing its performance on leaderboards like MS MARCO Document Ranking [10]. ColBERT (Contextualized Late Interaction over BERT) uses a "late interaction" architecture that represents a middle ground between efficient bi-encoders and accurate cross-encoders. Unlike cross-encoders that process query-document pairs jointly, ColBERT generates contextualized token-level embeddings for queries and documents independently using a BERT-based encoder. This independence allows for efficient pre-computation and indexing of document embeddings.

Relevance computation compares these embedding sets using MaxSim (maximum similarity) operation, aggregating maximum similarity scores for each query token across all document tokens. This token-level way of operating keeps the expressiveness of cross-encoders and keeps the efficiency of separate encoding. The approach balances semantic understanding of deep language models with efficiency for large collection search, making it, in our opinion, suitable for scientific document retrieval.

We used RAGatouille library (v0.0.9) [7] for ColBERTv2 deployment and experimentation. RAGatouille provides a high-level API for ColBERT, simplifying document indexing and search while managing dependencies like FAISS for efficient nearest neighbor search over dense embeddings.

Implementation (i.e., Indexing and Search) We indexed document abstracts from training qrels. Each document input concatenated title, up to three author names (if available), and abstract, with fields separated by `[SEP]` tokens (e.g., "title text [SEP] author1; author2 [SEP] abstract text"). This structured input provided richer contextual information, allowing ColBERT to potentially learn different representations for titles, authors, and abstracts. Documents were indexed using pre-trained `colbert-ir/colbertv2.0` weights via RAGatouille; no additional fine-tuning on LongEval-SciRetrieval was performed for this end-to-end experiment.

Maximum document length was 482 tokens (`MAX_SEQ_LENGTH - 30`, where `MAX_SEQ_LENGTH` was 512) (to try to speed up the process and reduce the computational load), with RAGatouille handling document splitting for longer texts. RAGatouille managed ColBERT index creation, encoding all documents into token-level embeddings and organizing them for efficient search using approximate nearest neighbor techniques. The code for this setup can be found in `pipeline/task_2/ColBERT_v2.ipynb`.

ColBERTv2 End-to-End Performance (Development Set) Pre-trained ColBERTv2 via RAGatouille for end-to-end retrieval (indexing and searching abstracts from our development partition) produced strong results, shown in Table 4. The nDCG@10 score of 0.6363 substantially outperformed both BM25 baseline and fine-tuned SciBERT, achieving 64% of ideal ranking quality. This represents a 3.4× improvement over SciBERT's nDCG@10, demonstrating ColBERT's superior semantic understanding. And the most important, without any fine-tuning nor using the "full-text" dataset, which definitely would have resulted in a better score.

The MAP score of 0.6458 indicates good precision across all relevant documents, while the MRR of 0.7928 shows consistent placement of relevant results in top positions. These results suggest that ColBERT's late interaction mechanism effectively captures scientific document semantics without, again, domain-specific fine-tuning. This can be due to its pre-training on diverse text and architectural advantages in handling long documents. The strong performance across all metrics indicates that ColBERT successfully is a big step on bridging the gap between lexical and semantic retrieval in this domain.

Table 4. Comparison: SciBERT vs. ColBERTv2 End-to-End.

Metric	SciBERT	ColBERTv2
nDCG@10	0.1896	**0.6363**
AP (MAP)	0.0999	**0.6458**
RR (MRR)	0.7564	**0.7928**

Challenges with Full-Text ColBERTv2 End-to-End Scaling end-to-end ColBERTv2 to the full-text document collection was infeasible under our project constraints. The main obstacle was infrastructure, the TU Wien's JupyterHub cluster was missing the NVIDIA CUDA Compiler (i.e., `nvcc`). This compiler builds optimized CUDA C++ extensions for libraries like `faiss-gpu` (an internal RAGatouille dependency) and ColBERT's custom kernels. A lot of efforts were made to install it in the cluster but there was no success.

The next option we tried was to migrate to Google Colab Pro, as it provided GPU access with the necessary CUDA toolkit, but introduced new limitations such as session runtime limits (exceeded when processing large full-text datasets) and costs from using high-end GPU usage (e.g., NVIDIA A100). These infrastructure constraints indeed can be used as an example to highlight the main challenge in (neural) IR methods: state-of-the-art models often require specialized hardware and extended computational resources, which in most of the cases may be inaccessible in a master student context. The 24 GB full-text **zipped** corpus requires a long indexing time and memory. This made our focus to shift on the abstract, smaller dataset to demonstrate in a practical way that ColBERT is a powerful option for scientific retrieval.

5 Approach 3: Two-Stage Re-ranking (BM25 + ColBERTv2)

The "retrieve and re-rank" is an effective strategy in modern IR systems. It adresses the computational trade-off between efficiency and effectiveness. This multi-stage approach comes from the fact that neural rankers are computationally expensive to apply to entire document collections, even if they bring better results (as we have seen in task 2). This new strategy combines the speed and coverage of traditional retrieval methods with the semantic sophistication of neural models. Usually this is done through two stages:

1. The first stage employs a retrieval model like BM25 which we explored in the first approach. These models are designed to have a high recall, and to identify a set of potentially relevant documents from the entire corpus.
2. The second stage then applies a more sophisticated, computationally expensive model, such as a neural re-ranker, like the ones we explored in the second approach. This neural re-ranker, re-evaluates and re-orders the first stage candidates, adding semantic understanding to improve precision at the top of the ranked list.

This approach was indeed our first submission strategy for the LongEval CLEF, as pure ColBERT [11], was not an option due to lack of resources.

5.1 BM25 First-Stage Candidate Retrieval

For the initial retrieval phase, we employed our BM25 implementation on document abstracts. Our text preprocessing pipeline (e.g., lowercasing, punctuation/digit removal, NLTK `word_tokenize`, stemming and English stopword removal) was applied. The BM25 parameters for this first stage were set to $k_1 = 1.5$ and $b = 0.75$. These values, were selected to get the best recall in the initial candidate set.

The choice of $k_1 = 1.5$ allows for greater term frequency influence, potentially capturing more documents with repeated query terms. The $b = 0.75$ value provides moderate length normalization, balancing between treating all documents equally and heavily penalizing longer abstracts.

For each query, the top $K_{BM25} = 200$ candidate abstracts were retrieved from the entire document collection (abstracts from the training data for development, and test set abstracts for final runs). Finally, token caching was also used for efficiency during test set execution.

5.2 ColBERTv2 Second-Stage Semantic Re-ranking

The set of 200 candidate documents retrieved by BM25 for each query was then passed to the second stage for semantic re-ranking. We used a pre-trained `colbert-ir/colbertv2.0` model, loaded via the RAGatouille library [7]. It is important to emphasize that for this re-ranking stage, the ColBERT model was employed in a *zero-shot* capacity. Its weights, pre-trained extensively on general corpora like MS MARCO, were used directly without any further fine-tuning on the LongEval SciRetrieval dataset's relevance pairs.

The input to the ColBERT re-ranker for each candidate document was constructed by concatenating its title, up to three author names (when available), and its abstract, using "[SEP]" as a separator token. This structured input allows ColBERT to leverage different fields distinctly, giving more weight to title matches or author relevance signals. RAGatouille's built-in methods were used, returning a new ranking based on ColBERT's fine-grained MaxSim computations.

From the re-ranked list of 200 candidates, the final top $K_{final} = 100$ documents were selected to form the output for that query. During execution of our re-ranking notebooks, we observed system warnings regarding duplicate document entries being passed to the re-ranker, a data characteristic that potentially affected the performance.

5.3 Implementation and Submission Workflow

The development, parameter selection (for BM25 in this stage), and performance assessment of this re-ranking pipeline were conducted using a Jupyter Notebook (`pipeline\task_3\evaluation.ipynb`)

for the training/development data and its associated qrels. Figure 3 highlights the architecture of this implementation.

For the official TIRA submissions, a separate, dedicated notebook was created (`pipeline\task_3\exec_and_uploader.ipynb`). This submission script processed both the November 2024 and January 2025 test query sets against the test document abstract collection.

For each query set, it performed the two-stage retrieval: BM25 candidate generation followed by ColBERTv2 re-ranking. The final ranked lists were then formatted into TREC-compliant run files (e.g., `run.txt`), which were subsequently compressed to `run.txt.gz` as per TIRA submission guidelines. The notebook also integrated `tira-cli` commands to facilitate the validation and upload of these submission packages to the TIRA platform.

Finally, before the deadline, we decided to try a new iteration of the code in a new notebook (`pipeline\task_3\evaluation_v2.ipynb`), trying to reuse as much code as possible from approach 1. However, the results were worse than the first iteration. In this section, the first iteration is documented.

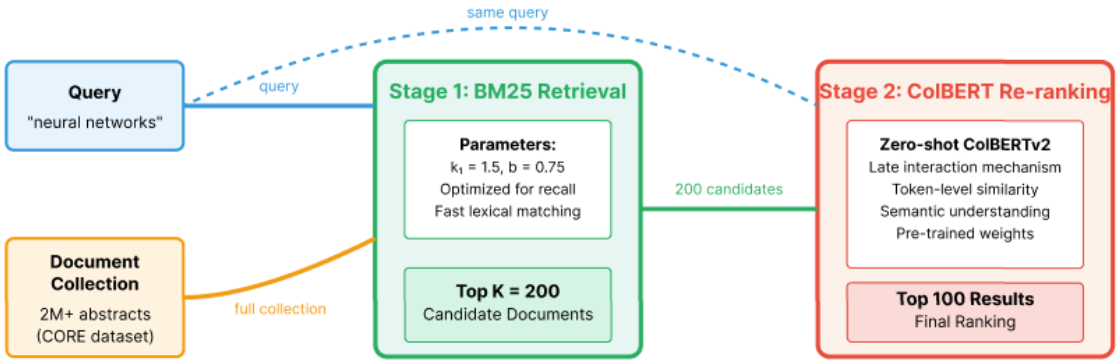

Fig. 3. Two-stage re-ranking pipeline combining BM25 for efficient candidate retrieval with ColBERT for semantic re-ranking.

5.4 Re-ranking Performance (Development Set)

On the development set (abstracts from the training data partition), the BM25 + ColBERTv2 re-ranking pipeline achieved the performance metrics summarized in Table 5. The nDCG@10 score of 0.1316 indicates that the hybrid approach achieved moderate ranking quality, though below the standalone ColBERT end-to-end performance.

The relatively lower performance compared to end-to-end ColBERT suggests potential limitations in the two-stage approach: the BM25 first stage may adds a "recall ceiling", preventing highly relevant but lexically dissimilar documents from reaching the re-ranking stage. Additionally, the different BM25 parameterization for recall optimization (rather than precision) may have introduced noise that the ColBERT re-ranker struggled to overcome within the limited candidate set.

Table 5. BM25 + ColBERTv2 Re-ranking Performance.

Metric	Score
nDCG@10	0.1316
AP (MAP)	0.1218
RR (MRR)	0.2717

The MAP score of 0.1218 and MRR of 0.2717 are lower than end-to-end ColBERT, suggesting that the two-stage approach may lose relevant documents in the first stage or that the re-ranking operates

on a candidate set that is too constrained by lexical similarity. This highlights a key challenge in hybrid retrieval systems: optimizing the interaction between retrieval stages to maximize the benefits of both approaches.

6 Overall Results and Discussion

Table 6 presents a comparative overview of the best nDCG@10 scores achieved by our main implemented approaches on the abstract development set. A visual chart can be seen in Figure 4.

Table 6. Overall nDCG@10 Performance Comparison on Abstract Dataset (Development Split).

Approach	nDCG@10
BM25 (Standalone, $k_1 = 1.0, b = 0.7$)	0.1778
SciBERT (Fine-tuned)	0.1896
ColBERTv2 (End-to-End, RAGatouille)	**0.6363**
BM25 ($k_1 = 1.5, b = 0.75$) + ColBERTv2 Re-ranker	0.1316

The end-to-end ColBERTv2 system (indexing with pre-trained weights) demonstrated superior performance on the development set, achieving 0.6363 nDCG@10. This represents a substantial improvement over both traditional lexical approaches (BM25) and domain-adapted language models (SciBERT). The success of ColBERTv2 highlights the effectiveness of late interaction architectures for scientific document retrieval, even without domain-specific fine-tuning.

The BM25+ColBERTv2 re-ranking system, despite lower scores (0.1316), was selected as our primary submission due to the execution feasibility compared with the resource constraints that prevented full-text ColBERT indexing. The performance gap between end-to-end ColBERT and the hybrid re-ranking approach suggests that the BM25 first creates a recall ceiling, preventing semantically relevant but lexically dissimilar documents from reaching the re-ranking stage. Additionally, the different BM25 parameterization optimized for recall rather than precision may have introduced candidates that the ColBERT re-ranker struggled to properly rank within the limited candidate set.

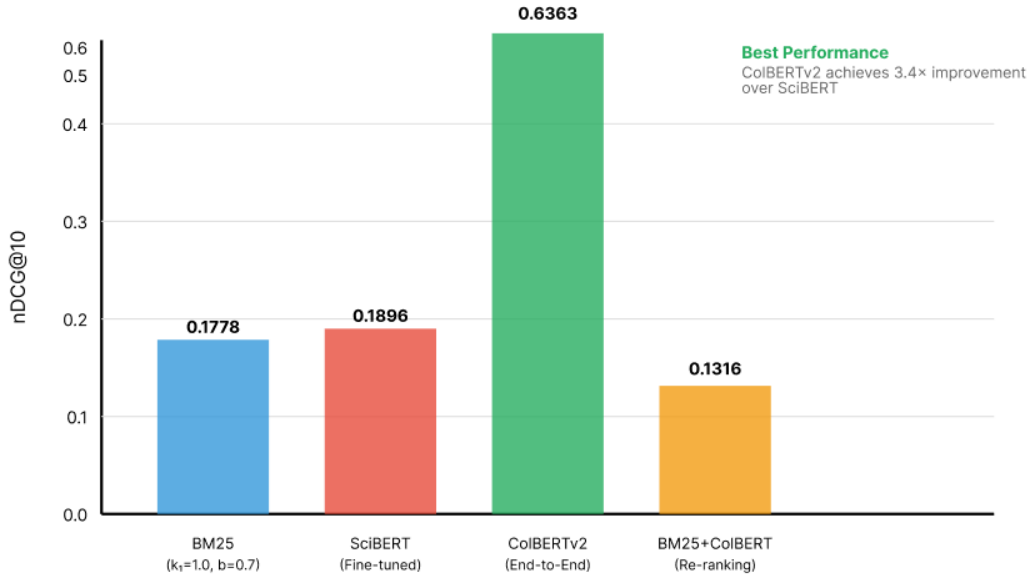

Fig. 4. Performance comparison of all approaches on abstract development set.

Our work faced several challenges that influenced our experimental design and submission strategy:

- **Initial Pipeline Complexity:** Our ambitious modular pipeline design proved too complex for the diverse experimental needs and team skill levels. We simplified to task-focused scripts and Jupyter Notebooks for faster iteration.
- **Computational Infrastructure:** The TU Wien JupyterHub cluster's lacked of `nvcc` (NVIDIA CUDA Compiler), which forced a migration to Google Colab Pro, incurring monetary costs and session time limits. This particularly affected full-text ColBERT experiments, which require substantial GPU resources and extended runtime.
- **Dataset Scale:** The large dataset size (60 GB full-text, 2M+ documents) created processing bottlenecks. BM25 tokenization was time-intensive, and neural model processing required careful memory management. Token caching and chunk-based processing helped mitigate these issues.
- **Resource Constraints:** Limited computational budget and time restrictions prevented extensive hyperparameter optimization and full exploration of the full-text dataset with neural models, potentially impacting final performance.

These challenges highlight the practical difficulties of conducting neural IR research in educational settings, where access to high-performance computing resources may be limited.

7 Own Contribution

Our primary contributions to the LongEval-SciRetrieval task include:

- **Systematic Comparison of IR Paradigms:** We conducted a comprehensive evaluation of lexical (BM25), neural end-to-end (SciBERT, ColBERTv2), and hybrid re-ranking strategies on scientific document retrieval, providing insights into their relative strengths and limitations.
- **Detailed BM25 Analysis:** Our hyperparameter analysis revealed document-type-specific optimal configurations, with grid search results that show the corpus-dependency of BM25 parameters. We also experimented with different BM25 implementations and variants and documenting.
- **Domain Adaptation Exploration:** We adapted SciBERT for scientific document relevance classification and showed ColBERTv2's effectiveness via RAGatouille on LongEval abstracts, previewing that late interaction architectures can generalize across domains.
- **Hybrid Pipeline Development:** We developed and evaluated a BM25+ColBERTv2 re-ranking pipeline that balances computational efficiency with semantic understanding, though the mixed results highlight the challenges of multi-stage retrieval.

8 TIRA Submissions and Leaderboard Status

Official leaderboards for the LongEval-2025 task were not available at the time of writing. We successfully submitted multiple runs to TIRA, including our primary `BM25_ColBERTv2_ReRanker_v1` and standalone BM25 variants as shown in Table 7.

Table 7. List of Submitted Runs to TIRA and Development Set nDCG@10 (Abstracts for ReRanker run, full-text for single BM25 runs.

Run Tag on TIRA	System Configuration	nDCG@10
BM25_ColBERTv2_ReRanker_v1	BM25 ($k_1 = 1.5, b = 0.75$) + ColBERTv2	0.1316
BM25_k1_0p2_b_0p8_stop_stem_fullText	BM25 (Full-text optimized)	0.2002
BM25_k1_1p0_b_0p7_stop_stem_fullText	BM25 (Abstract optimized)	0.1937
BM25_k1_0p95_b_0p75_stop_stem_fullText	BM25 (Default)	0.1959

Our submission strategy prioritized the hybrid re-ranking approach as our primary run, supported by several BM25 variants to establish baselines across different parameter configurations.

9 Conclusion

We explored three different IR strategies for the LongEval-SciRetrieval task, showing varying levels of effectiveness across lexical, neural, and hybrid approaches. End-to-end ColBERTv2 showed the highest potential on development abstracts (nDCG@10=0.6363), outperforming traditional lexical methods and domain-adapted language models. This showcases the effectiveness of late interaction architectures for scientific document retrieval, even without domain-specific fine-tuning.

Our submitted BM25+ColBERTv2 re-ranking system balanced performance considerations with computational feasibility under resource constraints, though with mixed results that highlight the complexity of optimizing multi-stage retrieval pipelines. The performance gap between end-to-end and hybrid approaches suggests that a hypothetical future work should focus on improving the interaction between retrieval stages or exploring alternative hybrid architectures.

Key challenges included computational limitations, dataset scale, and infrastructure constraints that cannot be avoided from a student perspective.

An hypothetical future work should target full-text ColBERT indexing with adequate computational resources, task-specific ColBERT fine-tuning on scientific literature, and optimization of hybrid retrieval pipelines to better use the benefits of both lexical and semantic approaches.

References

1. DBLP Computer Science Bibliography: Publications per Year Statistics. https://dblp.org/statistics/publicationsperyear.html (Accessed: May 26, 2025)
2. CLEF LongEval Lab Homepage. https://clef-longeval.github.io/ (Accessed: May 26, 2025)
3. Matteo Cancellieri, Alaa El-Ebshihy, Tobias Fink, Petra Galuščáková, Gabriela Gonzalez-Saez, Lorraine Goeuriot, David Iommi, Jüri Keller, Petr Knoth, Philippe Mulhem, Florina Piroi, David Pride, Philipp Schaer: LongEval at CLEF 2025: Longitudinal Evaluation of IR Model Performance. arXiv:2503.08541 (2025). https://arxiv.org/abs/2503.08541 (Accessed: May 26, 2025)
4. Chuklin, A., Markov, I., de Rijke, M.: Click Models for Web Search. Morgan & Claypool Publishers (2015). https://doi.org/10.2200/S00654ED1V01Y201507ICR043
5. LongEval SciRetrieval Task Dataset. TU Wien Research Data. https://researchdata.tuwien.ac.at/records/r643n-yc044?preview=1 (Accessed: May 26, 2025)
6. LongEval SciRetrieval Test Task Dataset. TU Wien Research Data. https://researchdata.tuwien.ac.at/records/v8phe-g8911?preview=1 (Accessed: May 26, 2025)
7. Gautier, B., et al.: RAGatouille: Library for using ColBERT in RAG applications. GitHub Repository. https://github.com/AnswerDotAI/RAGatouille (Accessed: May 26, 2025)
8. Beltagy, I., Lo, K., Cohan, A.: SciBERT: A Pretrained Language Model for Scientific Text. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 3615–3620. Association for Computational Linguistics, Hong Kong, China (2019). https://doi.org/10.18653/v1/D19-1371
9. Khattab, O., Zaharia, M.: ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), pp. 39–48. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3397271.3401075
10. Microsoft Research: MS MARCO Document Ranking Submissions Leaderboard. https://microsoft.github.io/MSMARCO-Document-Ranking-Submissions/leaderboard/ (Accessed: May 26, 2025)
11. Paul, D.: Revolutionizing Information Retrieval with RAG, Reranking, and Colbert. Medium Article. https://medium.com/@diptamay/revolutionizing-information-retrieval-with-rag-reranking-and-colbert-6069f195a733 (Published: Jan 2, 2024, Accessed: May 26, 2025)
12. Xiang Han Lù, BM25S: Orders of magnitude faster lexical search via eager sparse scoring. https://arxiv.org/abs/2407.03618 (Accessed: May 28, 2025)
13. Experimental implementation of BM25s https://github.com/mtiessler/188.980_Advanced_Information_Retrieval_Project/tree/main/deprecated/bm25s_experimentation