CLEF 2025 – CheckThat Subtask 4b
Approach of Team TUWien_AIR2025_G15

Kartik Arya
e12402369@student.tuwien.ac.at

Teodor Esaiasson
e12443590@student.tuwien.ac.at

Nooreldin Lasheen
e12302427@student.tuwien.ac.at

Daniel Levin
e12433760@student.tuwien.ac.at

Philipp Moeßner
e12412779@student.tuwien.ac.at

Code and Repository

The repository and source code for our final approach can be found here:
Submission GitHub Repository

The working files (also of the additional tested approaches) can be found here:
Working Files GitHub Repository

1. CLEF Challenge

Social media platforms such as X.com (formally known as Twitter) are mostly used for informal and fast-paced public discussions. When scientific topics are debated on these platforms, stated claims often lack transparency regarding their sources or general evidence. Therefore, fact-checking scientific discussions on social media platforms is of major importance. Task 4 of the CheckThat-Lab at CLEF 2025 takes on this problem, specifically for the COVID-19 domain. We participated in task 4b which asks its participants to develop an information retrieval system which outputs a ranked list of scientific documents for a given tweet making a statement about supposedly conducted research. The better the tweet's content matches a document's content, the higher it shall be ranked in the output list. In order to achieve this, the organizers provided a corpus of 7718 documents which are possibly mentioned in the tweets. Added to that, 12853 tweets and the corresponding correct document mentioned by the tweet were provided as training data. 1400 tweets (+ correct document) were provided as development data to internally test and evaluate the trained approach. Finally, 1446 tweets were provided as test data without the corresponding correct documents. The ranked lists for each of the test data tweets were then submitted to the challenge, evaluated by the organizers and served as the final result of a team on the leaderboard. The following report examines our approach for the demanded IR system.

2. General Approach

We split the given task into 3 sub-tasks: First (3), we built a traditional IR model, meaning an approach that is solely based on individual term frequency and (inverse) document frequency of terms, namely BM25. Secondly (4), we made use of representation learning approaches to build abstract embeddings for both documents and queries (tweets) to represent them in a more holistic and context-aware way. Finally (5), we used these embeddings as input for different neural reranking approaches, which rerank and potentially improve the BM25 preranked lists.

3. Traditional IR Model

3.1. BM25

From the organizers of the task, a BM25 model was given as a baseline. Since later reranking approaches are built on the initial documents retrieved by the BM25 model, this offered a starting point for improving the final results of the predictions.

3.2. What is BM25?

$$rsv_q = \sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) . \left(\frac{(k_1+1).tf_{td}}{k_1.\left((1-b)+b.\left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}} + \delta \right)$$

Figure 1: BM25 formula [1]

Best match 25 (BM 25) is a basic approach to information retrieval that is based on the concept of counting how rare terms are in the whole document corpus and how often the terms appear in any of the documents. To do this, an inverted index is created of all terms across all documents which contains information about the frequency of the term in a given document. To use BM25 in an information retrieval task, a score is calculated for each term (or rather token) in a query together with each document in the corpus. The score for all tokens in combination with that document is then summed up as the total score for that query document pair. In an intuitive sense, the score is based on the frequency of the token in a specific document multiplied by how rare the token is across all documents.

There are many variations of BM25, but the mathematical formula for the one used in this project can be seen in Figure 1.

3.3. Modifications and Contributions

Throughout the project, we iteratively improved the baseline BM25 model by addressing domain-specific challenges (a detailed documentation of the conducted iterations can be found here).
The first milestone was to create a good preprocessing function that would extract high-quality terms from queries and documents given their nature and domain. For that, we decided to let every member of the team get familiar with the BM25 baseline and see which approaches to text preprocessing each one can come up with, without influencing each other. After we had collected 5

results, we analyzed them and combined them into a single pre-processing function that became the core of the further improvement of BM25. The main challenges included the informal and noisy nature of tweets, the vocabulary gap between tweets and scientific documents, and the need to maximize retrieval effectiveness given the limited overlap in terminology.

Our next step was to clean and optimize the provided baseline code, introducing parallelization (using `pandarallel`) and caching to speed up processing. Recognizing the importance of domain-specific terminology, we experimented with various spaCy models for named entity recognition, including `en_core_web_md` and `en_core_sci_md`. However, we found that the choice of model had only a marginal impact on retrieval performance, with `en_core_web_md` providing the most stable results across samples, while the performance and memory footprint varied significantly.

To further bridge the vocabulary gap, we incorporated a COVID-19-specific thesaurus [2] to normalize synonyms in both tweets and documents. This synonym normalization, when applied before other pre-processing steps, did not yield improvements when combined with named entity recognition. However, omitting named entity recognition and relying solely on synonym normalization led to an increase in MRR, suggesting that heavily relying on named entities could introduce noise rather than signal.

We also evaluated the impact of retaining numerical information, which is often critical in scientific discourse. By preserving numbers during pre-processing, we observed a slight improvement in retrieval metrics.

Throughout the iterations we were aware of the trade-off between complexity and performance. We aimed for simplicity as long as the results weren't significantly affected. This was particularly important in the context of our project, with multiple members collaborating on the same model which needs to be maintained in a format of jupyter notebooks. Speed of running notebooks on each member's machine was an important consideration, as well as the readability of the code.

Finally, we compared different BM25 variants available in the `rank_bm25` library, including BM25Okapi, BM25L, and BM25Plus. Our experiments showed that BM25Plus consistently outperformed the other variants on both the training and development sets, likely due to its improved handling of term frequency saturation in relatively short documents. At the same time, we observed that BM25L performed very poorly which was a surprising result. Although it is more suitable for longer documents, we did not expect it to output such a low score compared to BM25Okapi and BM25Plus.

3.4. Final Model and Performance

Our final BM25 pipeline combined several of the most effective strategies identified during the iterative process that was based on the findings of each member of the group. We used the BM25Plus model with synonym normalization (using the COVID-19 thesaurus [2]), careful text cleaning, and retention of important numerical terms. Named entity recognition was omitted, as it did not provide additional benefits in this context.

This approach resulted in a substantial improvement over the original baseline. As summarized in Table 2, our best BM25 configuration achieved an MRR@5 of 61.76% on the development set, compared to 55.20% for the baseline. The improvements can be attributed to domain-specific synonym normalization, optimized preprocessing, and the selection of the most suitable BM25 variant for our data.

Our contributions lie in the systematic evaluation of preprocessing strategies, the integration of

external domain resources (thesaurus [2]), and the empirical comparison of BM25 variants. These steps addressed the core challenges of vocabulary mismatch, style gaps between queries (tweets) and documents (scientific articles) and noisy input, enhancing the retrieval effectiveness of the traditional IR model.

4. Representation Learning

To not only rely on exact word / token matching like in our traditional IR model, we transitioned to token embeddings for the reranking approach. There are already a lot of pretrained models for embedding creation that also roughly fit our domain. Also, the challenge did not provide enough training data to train a state-of-the-art transformer model from scratch. We therefore chose a transfer learning setup and finetuned a pretrained BERT instance.

4.1. BERT Finetuning

BERT model (in ColBERT)	MRR@5 (dev)
SciBERT[1]	68.03%
BioBERT[2]	67.56%
PubMedBERT[3]	67.05%

Table 1: Performance of different BERT versions in the ColBERT reranking architecture

To find a BERT model which works best for our domain we tried out different public versions from Huggingface such as SBERT, MiniLM, SciBERT, BioBERT or PubMedBERT and evaluated their baseline effectiveness.
Within the MiniLM family, we compared the 12-layer (multi-qa-MiniLM-L12-cos-v1) and the 6-layer distilled version (multi-qa-MiniLM-L6-cos-v1). Interestingly, the 6-layer model outperformed the larger model in terms of MRR, highlighting that smaller, distilled models can be more efficient than larger ones especially when the complexity of the dataset does not need deep contextual representation.
Additionally, we tested input preparation methods. Here, we encountered items in the collection, particularly abstracts and titles, that exceeded the 512-token input limit of BERT-based models, sometimes reaching up to 1372 tokens. We tried truncating these inputs, but this did not improve performance and, in some cases, led to a loss of relevant content. We also experimented with standard preprocessing techniques such as lowercasing, punctuation removal, and stopword filtering. These had varying effects across models: While a cross-encoder MiniLM benefited from preprocessing, SBERT's performance declined, as an unexpected result given the importance of contextual understanding in such models. This illustrates that each model has its own way of handling input and preprocessing. Some approaches enhanced MRR for certain models, while the same methods led to reduced MRR in others.

[1]Huggingface: `allenai/scibert_scivocab_uncased`
[2]Huggingface: `dmis-lab/biobert-v1.1`
[3]Huggingface: `microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract`

After these first tests in a lot of different directions within different reranking approaches we focused on our best performing reranking approach ColBERT (section 5). We again specifically tested it on different domain-specific pretrained BERT instances. We thereby focused on BERT models that cover the biomedical scientific domain instead of e.g. Twitter-specific BERT models since only the queries (which are tweets) would benefit from such a BERT model while a more general biomedical specific model can cover both the documents' and tweets' content. Table 1 summarizes our results and reveals that SciBERT fits ColBERT reranking and the task's data best.

5. Neural Re-Ranking

System	MRR@5 (dev)
ColBERT	68.03%
Cross-encoder	49.46%
MatchPyramid	42.01%
Conv-KNRM	36.28%
Transformer Kernel	0.034%
BM25 (modified)	61.76%
BM25 (baseline)	55.20%

Table 2: Best achieved MRR@5 scores on the dev dataset for different tested reranking approaches compared to BM25 preranking

5.1. ColBERT

We achieved the highest MRR@5 score on the development dataset with a ColBERT reranking architecture (Table 2). Its reranking mechanism is quite simple compared to other approaches (Section 5.2). Queries and documents are both passed through the same encoder resulting in token-level embeddings for each query and each document. To compute a relevance score of a document for a given query, a match matrix is built consisting of cosine similarity values for each query token – document token pair. The final score is aggregated from the matrix by simply summing up (over query dimension) the maximum values of each row (document dimension).

5.1.1 Modifications and Contributions

As mentioned in section 4 we used a SciBERT instance as encoder for both queries and documents after finetuning it to our specific domain (COVID-19 related tweets and documents). This finetuning is the only part of ColBERT training since the scoring / reranking mechanism of ColBERT does not have to be trained as it is a simple matrix aggregation rule. Still, in the context of ColBERT, the SciBERT finetuning loss function is adjusted: The training goal is to obtain a higher score for a relevant than for a non-relevant document while satisfying a predefined margin between the scores. These scores are already computed as the ColBERT-specific match matrix aggregation instead of e.g. simple cosine similarity of CLS vectors.

Finetuning epochs	MRR@5 (dev)
0	56.94%
2	63.51%
6	**64.02%**
10	62.26%

Table 3: Snippet from finetuning SciBERT (ColBERT reranking) for a different amount of epochs

To adjust ColBERT to the challenge's task, we tuned its hyperparameters as follows: First, we tried out training for different amounts of epochs and found that training for 6 epochs gave the best MRR@5 score on the dev dataset (table 3). Two additional important training parameters are the number of negative document samples for a query and the margin. We performed a grid search (with a fixed number of epochs at 6 and a learning rate at 2e-5) to find the best combination of these parameters and found that a margin of 0.5 and 1 negative sample achieved the highest MRR@5 score (table 4).

Margin	Negatives	MRR@5 (dev)	Last Epoch Loss
0.3	1	0.6537	8.0212
0.3	2	0.6477	16.7373
0.3	4	0.6118	29.3307
0.4	1	0.6574	12.9181
0.4	2	0.6503	21.4273
0.4	4	0.6360	27.8490
0.5	**1**	**0.6610**	**9.6190**
0.5	2	0.6393	18.1380
0.5	4	0.6259	31.5070

Table 4: Grid search for finding best margin and number of negative samples (table has cutoff at margin 0.5 since we saw MRR@5 go down again with higher margins)

Lastly, we checked whether the length of the BM25 preranked lists influence the MRR@5 score and found that a length of 100 documents results in the best score (5).

Preranked List Size	MRR@5 (dev)
25	67.75%
50	68.03%
100	**68.06%**
200	67.58%

Table 5: MRR@5 on dev set for different preranked list sizes (number of documents) using ColBERT (SciBERT) with margin 0.5

5.1.2 Evaluation

Overall, the ColBERT reranking approach increased the MRR@5 (dev) score by almost 13% points compared to the BM25 baseline and around 6% points compared to our modified BM25 model. None of our other tested approaches achieved this increase. Thus, we used this ColBERT configuration (trained in 6 epochs with margin 0.5, 1 negative sample and a learning rate of 2e-5) in our ranking pipeline for the test data and our final submission to the leaderboard.

5.2. Other Tested Approaches

5.2.1 Cross-encoder

One of the attempted approaches was a cross-encoder approach, which ultimately did not turn out to offer improvements compared to the ColBERT approach described previously. The basic idea of cross-encoding is that every query and document is processed together, and the CLS-token is computed as a combination of the query and document. This is a major disadvantage in a production environment since it means that the CLS-token for documents in the corpus can not be precomputed, causing problems in a situation where the results needs to be presented quickly. The processing time is however not one of the major concerns in this task and therefore a cross-encoding approach could have been an advantageous approach in theory since it can better caption the relationship between query and document in comparison to approaches where embeddings are computed separately.
Two different BERT models were tested using cross-encoding, SciBERT and PubMedBERT, as well as one of the MiniLM variety, pre-trained on the MS Marco dataset. The difference was also that the MiniLM model had been pre-trained with the cross encoding approach.
As can be seen in (6), the two BERT models performed poorly on this approach, while the MiniLM model had a significantly better result. We chose to go in the direction of the ColBERT approach relatively early in this process and therefore did not go further in testing if the results could possibly be improved when using a BERT model. These results nonetheless show that cross-encoding can not be applied to any pre-trained model, and there is a need of pre-training models using cross-encoding, to be able to use them for this purpose.

Model	MRR@5 (dev)
SciBERT	0.00%
PubMedBERT	2.94%
ms-marco-MiniLM-L-6-v2	49.46%

Table 6: Performance of different models using a cross-encoding approach

5.2.2 Conv-KNRM

One more approach we have decided to try out is Conv-KNRM (Convolutional Kernel-based Neural Ranking Model), the most effective model that was presented in the lecture about Neural Re-Ranking approaches. This neural architecture leverages kernel pooling to capture diverse similarity signals between query and document terms and borrows the convolutional approach from

image processing to be able to capture higher-order relationships between textual tokens. In the scope of this course, we have decided to focus on lighter variations of KNRM architectures, i.e. plain KNRM without convolutional layers, KNRM with 1-grams, and KNRM with bigrams and 8 convolutional channels. The training time grew linearly with N (of N-gram) which was the main reason we decided to stop at bigrams. Specifically, we defined similarity buckets centered at $[-1.0, -0.5, 0.0, 0.5, 1.0]$, allowing the model to effectively distinguish between various degrees of semantic relatedness, from strong negative correlations to exact matches.

For token representations, we leveraged a pretrained ColBERT model, which has proved itself as a reliable choice for this task, both in terms of accuracy and performance. This was the optimal balance between effectiveness and computational efficiency. The 1-gram model showed a very low MRR@5 score. It was hard to find an explanation for this phenomenon, which generally was observed while trying out other neural network architectures with low explainability. After having tried multiple approaches for information retrieval we came to the conclusion that simplicity and explainability of the models can sometimes prevail in the trade-off between effectiveness and maintainability.

As reflected in (7), the best-performing variation evaluated with MRR@5 on the dev dataset was the simplest one - KNRM. As for the convolutional variations, the training time was much longer than for the simple KNRM, and tuning of training parameters (e.g. mus, sigmas, epochs) brought no significant improvements to the MRR@5 score. Debugging of these models was constrained by their poor explainability and slower training time, even on the cluster.

Model	**MRR@5 (dev)**
KNRM	36.28%
Conv-KNRM with 1-grams	3.08%
Conv-KNRM with bigrams	2.83%

Table 7: Performance of different neural network architectures using KNRM approach

5.2.3 Transformer-Kernel Reranker

Inspired by recent interaction-focused architectures, we implemented ‘TKReRanker‘, a lightweight neural reranker built on a pretrained SciBERT encoder. At inference, every query and candidate document is first tokenized and passed through SciBERT to produce contextual token vectors. We then compute a full cosine-similarity matrix between the query’s and document’s tokens and pool these scores through 21 fixed Gaussian kernels (μ evenly spaced from –1 to 1, σ: constant). The resulting 21-dimensional kernel activations succinctly capture the distribution of similarity strengths across the token pairs. A small learned linear layer on top of these activations produces a single relevance score.

Training is performed end-to-end with margin-ranking loss (margin = 0.5) on triplets consisting of (tweet, positive document, negative document), where negatives are sampled from the BM25 top-100 shortlist. We fine-tune all of SciBERT together with the kernel head for 2 epochs using AdamW (learning rate 2×10^{-5}, weight-decay 0.01) and a linear warm-up over the first 10 % of total update steps.

Epoch	MRR@5 (dev)
0	0.0331%
1	0.0336%
2	0.0336%

Table 8: Development-set MRR@5 progression of the Transformer-Kernel reranker.

5.2.4 MatchPyramid

Another deep learning-based reranker that leverages interaction-based matching is the MatchPyramid model. It employs a convolutional neural network (CNN) to learn patterns from similarity matrices generated between token-level SciBERT embeddings of queries and documents. Unlike ColBERT or cross-encoder approaches, MatchPyramid treats the similarity matrix as an image and applies hierarchical convolution and pooling layers to extract relevance signals, followed by a fully connected layer to produce a final score. We trained the model from scratch using a margin-based triplet loss, with hard negatives sampled from the top BM25 results.Training with different epochs and learning rates showed best performance at 7 epochs with a 1e-4 learning rate, using input filtering to exclude noisy or short sequences.Despite these efforts, the MatchPyramid model underperformed relative to other rerankers. It achieved an MRR@5 of **42.01%** on the development set, which, while better than random, was significantly lower than the performance of our ColBERT-based approach. This result indicates that although MatchPyramid can capture local interaction patterns effectively, it may require a much larger training corpus or further architectural tuning to reach the performance levels of transformer-based models in this domain.

Epochs	Learning Rate	MRR@5 (dev)
3	1e-4	36.48%
5	1e-4	39.72%
7	**1e-4**	**42.01%**
9	1e-4	40.56%
7	5e-3	40.03%

Table 9: MatchPyramid reranker: MRR@5 on the development set for different epochs and learning rates. Best result achieved with 7 epochs and learning rate 1e-4.

6. Final Results

We ran our final ranking pipeline (modified BM25 preranking + finetuned SciBERT representation learning + ColBERT reranking) on the test data and submitted its output ranked lists per query to the challenge organizers. As a final result on this data we achieved a score of 0.58 MRR@5 making our group rank 14th place (of 31) on the leaderboard. This result marks a drop of over 10% points MRR@5 compared to the result we achieved on the dev data which suggests that the test queries were significantly different to the train and dev queries.

7. Discussion

Interestingly, the most "light-weight" approach in terms of necessary training effort performed best for the given task. A reason for that could be, that ColBERT only required fine-tuning (of SciBERT) and no from-scratch training like for example the CNN approach. Since the challenge organizers did only provide a very limited amount of training data it seems reasonable that approaches which heavily rely on a lot of training data performed worse. With more time we would have liked to improve our approaches, particularly the best performing approach ColBERT. Here, we could have tried another transforming it into a dual encoder setup where queries are encoded by a Twitter specific BERT instance and documents by SciBERT. Another finding we discovered while solving this task is: the more difficult the architecture of a model, the lower performance it shows and the harder it is for us to debug and explain the code to each other. This would be an important consideration for production models.

References

[1] A. Trotman, A. Puurula, B. Burges, *Improvements to BM25 and Language Models Examined*, November 2014, https://doi.org/10.1145/2682862.2682863 (accessed May 20, 2025).

[2] Patricia Fener, *COVID-19 Thesaurus*, https://loterre.istex.fr/C0X/en/ (accessed May 20, 2025).