Argument retrieval for comparative questions

Morotti Daniele, Sciamarelli Andrea and **Valente Andrea**
Master's Degree in Artificial Intelligence, University of Bologna
{ daniele.morotti, andrea.sciamarelli, andrea.valente6 }@studio.unibo.it

Abstract

The objective of this task, chosen among the ones of Touchè 2022[1], is to retrieve from a corpus of text passages the most relevant ones with respect to a given query. After that, a secondary task was to detect the stance of the text passages considering the objects in the query. Since we were not granted the access to the ChatNoir API (Bevendorff et al., 2018), which would have been very useful in order to retrieve the documents and, at the same time, several scores, we built our own indexes considering the available corpus. We decided to test different types of indexes: sparse, dense, and two hybrid approaches. The first one combines the scores of the sparse and dense indexes, the other one computes a re-ranking of the documents retrieved by the sparse index with MonoT5 (Nogueira et al., 2020). We reach our best result with the sparse retrieval and the MonoT5 re-ranking. For the stance detection we have to deal with unbalanced classes and BERT-like classifier.

1 Introduction

Document retrieval, or more generally "information retrieval", is a very well known task and it is used extensively in real applications, such as web search engines. The goal is to find the most relevant documents, in a large collection, that are related to a given query. Traditionally, retrieval systems leverage lexical similarities to match queries and documents using, for instance, inverted indexing techniques such as TF-IDF or BM25 weighting (Robertson and Zaragoza, 2009). These approaches, based on exact matches between tokens of the queries and documents, suffer from the lexical gap and do not generalize well (Berger et al., 2000). By contrast, approaches based on neural networks allow learning beyond lexical similarities, resulting in the state-of-the-art performance on datasets such as MS-MARCO. The last approaches introduced the use of Transformers, usually BERT-like models (Nogueira and Cho, 2019), which encode queries and documents together and can be either used to retrieve in an end-to-end manner the top-K documents through a k-nearest neighbor (KNN) search on the embeddings, or perform a re-ranking on the documents retrieved by a traditional system. Re-ranking architectures, while effective, exhibit high query latency because they perform expensive neural inference at query time. Thus, recently, there's been a growing interest in approaches based on representation learning, that allows document representations to be pre-computed independently from queries and stored off-the-shelf, thanks to which the search over the space of "query and document" vectors can be performed much more efficiently with indexes. These approaches are usually less effective than using re-rankers but far more efficient by offline indexing.

Another fundamental part of the task at hand is the one of "stance detection". Given a query - in our case a comparative question between two objects - and a document, we want to determine the "stance" of the document with respect to the query. Since the query is a "comparative question" between two objects, the possible stances are: "No", when the answer does not express a stance, but only some facts; "Neutral", when the answer does not favour particularly any of the two objects; "Pro 1^{st} object", when the answer favours the first object out of the 2 options; "Pro 2^{st} object", when the answer favours the second object out of the 2 options. The task is of major importance since, without a systematic method to determine the stance of a document, we wouldn't be able to perform the bigger task at hand as well. The general topic of "stance detection" is quite studied on the NLP community, however the materials available on the literature

[1] https://touche.webis.de/clef22/touche22-web/argument-retrieval-for-comparative-questions.html

drop drastically when looking for "stance detection" for "comparative questions". Nevertheless, the general consensus is that any suitable method should be able to capture semantic information and go beyond the superficial and usually insufficient lexical structure of the input which, in our situation, is the query and the document.

Taking into consideration the document retrieval and ranking, we tested different pipelines: a traditional sparse index based on BM25, a dense index that performs KNN on the embedding vectors of the documents, a hybrid approach that mixed up the scores obtained from the sparse and the dense index and a re-ranking model that retrieves the documents with the sparse index and then re-rank using the neural model Mono-T5. We started from the simplest models and then we tried to improve the results considering more sophisticated procedures. Instead, as far as the stance detection is concerned, we tested different combinations of networks and parameters. In particular, we first tested the performances obtained by just a single network, and then, after realizing that the performances were particularly poor, we tried a dual network architecture, taking inspiration from the paper (Bondarenko et al., 2022a), and finally we tweaked some parameters in order to improve even more the performances in terms of the considered score metrics (precision, recall, f1-score).

2 Background

After researching the task of "ad-hoc document retrieval", in the specific setting of "comparative questions", we realized that the literature covering the topic is not very rich, which gave us little to no material to take the inspiration from(Cai et al., 2021). However, after having consulted surveys and reviews of different approaches, we concluded that there is not a single paradigm which works empirically better in all tasks. The traditional methods usually achieve high recall efficiently but they lack in the ranking, while the neural methods are more difficult to implement but they can reach better results, even if sometimes they do not overcome statistical methods.

Transformers, as in other tasks of Natural Language Processing, change the entire paradigm and even future research directions in the domain of Information Retrieval, but they did not replace all available methods. Instead, their effectiveness is noticed in some parts along the pipeline, which for our case includes both retrieval and ranking.

In the retrieval approach we usually deal with a large corpus of documents, therefore there are some term-based models which work more efficiently compared to neural models. However, we discovered that the former models suffer from the so called "vocabulary mismatch", underlying the heterogeneity in length between queries and documents. In fact, queries often have a short and compact form, which express the question in terms of comparative arguments in just few words, whereas documents consist of many sentences but only some parts express the content we may be interested in. For this reason, neural methods, and in general architectures which overcome the limits of symbolic representations, are preferred. The given corpus of the Touché 2022 task, differently from previous years, is based on passage texts instead of documents, hence the mismatch depicted before is not relevant because the length of each passage text is way more smaller compared to the entire documents, thus term-based retrieval models could perform better than what we initially thought.

For what concerns "stance detection", even though, as mentioned, there is not a lot of literature available when it comes to perform it in the specialized realm of "comparative questions", we observed that we needed to go beyond the lexical information contained in the input and that, in particular, we needed to capture semantic information, and ground our decisions based on meanings. This can be done, for example, by manipulating what are called in the literature "contextual embeddings". Nowadays, we know that the architectures that shine when it comes to generating such "embeddings" able to capture semantic information are the Transformers-based architectures, such as the "Bidirectional Encoder Representations from Transformers", also known as "BERT". For this reason, we decided to approach the problem by defining a network based on (a variation of) a "BERT" architecture, as described in the following sections.

3 System description

Regarding the document retrieval and ranking stages, we took inspiration from several architectures already available in the literature. The goal was to retrieve a number of documents to subsequently rank using different approaches, i.e. BM-25 metric, late interaction of the ColBERT model, Mono-T5 re-rank(Pradeep et al., 2021) and finally

an hybrid approach involving both BM-25 and ColBERT.

The simplest model consists in retrieving `K` documents from the sparse index and computing the ranking considering the BM25 score.

The term-based retrieval method represents documents and queries using a bag-of-words, which improves efficiency but does not capture semantics and word ordering. The ranking is based on the BM25 model, which improves upon inverted indexing techniques like TF-IDF by addressing issues with term saturation and document length. BM25 uses the parameters k and b to control term saturation and the length penalty for longer documents (see equation 1).

$$BM25(D,Q) = \sum_{i=1}^{|Q|} IDF(q_i) \frac{f(q_i,D)\cdot(k+1)}{f(q_i,D)+k\cdot(1-b+b\cdot\frac{|D|}{avgdl})} \quad (1)$$

In order to enhance the baseline performance, we decided to create the sparse index on the expanded corpus (see Data), to observe if, by doing so, the model improves in terms of the relevance of the retrieved documents. The second tested pipeline makes use of embedding vectors and a dense index. First of all, we computed the embedding vector for each document using the Hugging Face library and importing the second version of "TCT-Colbert", pre-trained on MS-MARCO.[2] By computing the embeddings, we captured semantic information and relations between words, that can be exploited afterwards during the similarity search. In this pipeline, given a query, we compute its embedding and then we perform a KNN search over the space of all vectors, considering the inner product between them. The system returns the `K` documents with the highest inner product, which we interpreted as the ranking.

We decided, as third pipeline, to combine the scores from both the sparse and dense indexes, in order to capture the best documents retrieved by using the two described approaches. This model is a little more complex than the previous ones, as it needs to weight correctly the different scores, in order to avoid discarding relevant documents. The first step is to retrieve `K` documents from the sparse index and `K` from the dense index, where `K` is larger than the one specified in previous pipelines, subsequently we need to execute the algorithm to combine the scores, which consists of storing the maximum and the minimum values of both scores in order to use them later on. In the second step we compute the union between the IDs of the retrieved documents, then, for each document, if it was found by both indexes we take the actual scores, otherwise the minimum value (e.g. if a document was found only by one index we will take the actual score for that index but the minimum value for the other). After that we normalized the scores, but in our case the relevance metric got worse, and we summed the scores considering a parameter `alpha`, used to give more weight to one of the two scores. By default, the parameter `alpha` multiplies the BM25 score, because the ranking obtained by the dense index search on the embeddings is more reliable, nevertheless this behaviour can be inverted by passing a specific parameter. At the end, the system returns the new ranking, sorting the document IDs by the new scores. The algorithm was implemented in a similar way to the `HybridSearcher` class of Pyserini.[3]

The last pipeline that we implemented computes the re-ranking, using Mono-T5, on the documents retrieved by the sparse index. Mono-T5 follows the same principles of another re-ranking neural methods, i.e. MonoBERT, but adopts its strengths in a more powerful sequence-to-sequence approach. Firstly, it is a fine-tuned version of the T5 transformer model, which produces as output two tokens: true or false, depending whether each pair of query-texts are relevant to each other. Then, at inference time, it computes the pointwise re-ranking between each pair of query-passage text, where the output is the softmax distributions between only the "true" and "false" tokens. Finally, the probability of the "true" token is considered as the new score for each pair. Mono-T5 has gained popularity in several tasks and it is applied especially in the last stages of the retrieval-ranking pipeline, because of its effectiveness with a limited set of documents. Returning to our previous discussion, the idea is similar to the one of the hybrid approach: we would like to grasp the positive insights from both the sparse index and a more complex model. The Mono-T5 is available in Hugging Face[4], however it is also implemented in the PyGaggle library, developed by the same author of Pyserini, hence we proceed with the clone of their library repository

[2] Hugging Face: TCT Colbert v2 hnp MS-MARCO

[3] Github: Pyserini HybridSearcher class

[4] Hugging Face: Mono-T5 base

(the latest version is not available in the Python package manager pip). After having retrieved the documents from the sparse index, we simply passed the query and the text passages to the model, in order to compute the re-ranking. This model turned out to be very powerful and allowed us to get our best results in terms of nDCG@5.

As far as "stance detection" is concerned, as already mentioned, we agreed on the fact that the best option for tackling the task at hand should have been based on some type of BERT architecture, for the briefly discussed reasons. In particular, given the scarcity (quantitatively speaking) of the available labeled data, we went for a "Distilled" version of BERT, whose main property is being lighter and cheaper than the usual BERT architecture thanks to the reduced number of layers (Sanh et al., 2019). To be more precise, we used a pre-trained DistilBERT based architecture[5], to which we appended and trained a simple classifier simply made of `Dense` Layers, in order to make the network predict a class (in our case, the stance of a document with respect to a given query) after having encoded the input. We quickly realized that a single network wouldn't cut it as well as we'd have liked, therefore we consulted the literature and, inspired by the approach discussed in (Bondarenko et al., 2022a), we split the network in two different ones - the first detecting if the document was favouring an object or not, and the second determining whether it was favouring the first rather than the second object - we could have obtained better results. We actually verified that, in our situation, it did improve the performances. However, we noticed that, by taking into account the fact that, in our data-set, the distribution of classes was not uniform, we could have refined the networks even more in terms of quality of predicted classes during inference. For this reason, we decided to give different weights to each class so that, during training, each class was given more importance, proportionally to how under-represented it was in the data-set. After having trained independently both networks, we defined a small pipeline in order to yield a single output from both networks: given a query and a document, we predicted the stance with the first class and, only if the result turned out to be "Pro object", then we queried the second network with the same input to determine "which object". Effectively, this gave us a tool that, given the input, returned the same output of the first implemented network. For this reason, we were able to acquire evidence in favour of the fact that, by splitting the networks, we obtained a big boost in performances (qualitatively speaking).

[5] Hugging Face: DistilBERT base uncased mnli

4 Data

Data given by the Touché task are essentials for evaluations and further improve our performances. Moreover, we used a specific dataset for stance classification, which is always used in the literature when dealing with this type of task; it consists of 956 questions and answers taken from StackExchange[6] and Yahoo Answers[7], which were manually labeled with four stances: "No", "Neutral", "Pro 1^{st} object" and "Pro 2^{st} object" (see Introduction). Unfortunately, as mentioned multiple times, the labels distribution is unbalanced. In particular, only 69 records were labeled 0 (No stance), reason for which, without taking some kind of countermeasure, our system hardly labeled any input as 0, even when it rightfully had to do so. Instead, the remaining possible labels have a quite consistent support in the data-set: in particular, we counted 287 records labeled as "1" (Neutral stance), 324 records labeled as "2" (Pro 1^{st} object) and, finally, 276 records labeled as "3" (Pro 2^{nd} object), reason for which, in those situations, our models didn't struggle as much in detecting the correct class.

Furthermore, the most important data is the corpus, which is fundamental for our task since we have not been granted the permission to use the ChatNoir API [8] and we needed a set of documents for the retrieval stage. The corpus has 868655 passage texts. There's also another version of the same corpus, made up of the same passage texts expanded with generated queries. The difference has been crucial for our chosen architectures, thus a detailed discussion of the results has been done in section 6. For the expanded corpus, each query has been generated using the docT5query model (Nogueira et al., 2019).

Moreover, other important data are topics and corresponding judgements available for the evaluation of our pipeline. Judgements are needed for relevance, quality and stance assessment, each containing 2107 scores. For relevance and quality the possible values for each pair of passage

[6] https://stackexchange.com/

[7] https://it.wikipedia.org/wiki/Yahoo!_Answers

[8] https://www.chatnoir.eu/

text and topic are $0, 1, 2$, where 0 stands for low relevance/quality, while 2 is high relevance/quality. The stance has four possible values: NO, NEUTRAL, PRO 1st OBJECT, PRO 2nd OBJECT, which have been already discussed previously. The topics consist of 50 comparative questions, which we use to evaluate our pipeline on available judgements.

In addition, we noticed that many participants (Bondarenko et al., 2022b) preferred to use also the data from the task of 2021. The choice, however, is subjective and according to our evaluations we chose to use only the data from the current year. In fact, 2022 has been the first year which does not rely exclusively on ChatNoir and it does also provide a corpus at the passage level, instead of giving only the documents. Instead, considering the merge with last year data we should have created the indexes at document-level, hence increasing the complexity whilst lowering the effectiveness of the system. It is imperative to consider that the average length of passage texts is 900 characters, while for documents it grows up to 23430 characters. That is because, on average, each document includes around 26 passage texts. For these reasons, the disadvantages outweigh the advantages and thus we proceeded only with the corpus of this year.

Finally, the last dataset is a subset of MS MARCO [9] with comparative questions to perform passage-level relevance judgements, but it is useful only under some circumstances, such as training a neural-based retrieval system, which did not occur during our work and design choices. For example, we preferred to use the pretrained TCT-Colbert-v2 on MS MARCO, also considering our limited computational capabilities.

5 Experimental setup and results

In order to reproduce the experiments that we did for document retrieval, you should build the indexes on the corpus. To avoid this computationally heavy operation we left a link to a Google Drive shared folder in the notebook, such that you can download all the necessary datasets and indexes for running our code. In case you would like to build the indexes from scratch, you have to run Pyserini (Lin et al., 2021) for the sparse index and autofaiss for the dense one.[10] Moreover, if you would like to recreate the dense index you first have to create the embeddings, running inference on the entire corpus. As stated before, we discovered that the best results for the sparse index are obtained considering the extended corpus, moreover, you also have to set $k = 1.55$ and $b = 0.8$ for the BM25 function. We found these parameters by performing a simple grid-search over a narrow set of values. The best scores we obtained for the baseline pipeline, the sparse index built on the extended corpus, are $nDCG@5 = 0.508$ for the relevance and $nDCG@5 = 0.549$ for the quality.

Furthermore, we decided to test the RM3 query expansion method on this sparse index as well. The goal of the query expansion is to reformulate the query in order to improve the effectiveness of the search. In our case, we tested different parameters but without being able to get a better result than the basic sparse index. We suspect that the entire process is affected by "query drift", i.e. the model adds non-relevant documents, driving away from the original intent of the query.

Considering the pipeline that simply exploits the dense index, we did not have to test several parameters, because after we computed the embeddings and created the index there is not much more to do. We observed that the model got a better score if we clean the queries by removing the punctuation symbols before computing their embeddings. In this way we were able to obtain $nDCG@5 = 0.594$ for the relevance and $nDCG@5 = 0.614$ for the quality.

For the third pipeline, we already explained in section 3 how to combine scores and the main idea behind the implementation. However, during the tests, we noticed that by considering the RM3 query expansion, the scores of this model got worse. A possible explanation is that, in the algorithm, we take the union of the `K` documents retrieved by the sparse and the dense indexes, using RM3 we are introducing some sort of noise by adding some documents that are not properly relevant and therefore it affects the performance of the entire model. We noticed that the pipeline has better results when retrieving 700 documents in both indexes, and for the sparse index we set $k = 0.9$ and $b = 0.6$. The best weighting of the scores is obtained when setting $alpha = 0.2$, in order to give less importance to the sparse scores. With this setup we obtained $nDCG@5 = 0.619$ for the relevance and $nDCG@5 = 0.656$ for the quality.

The last model is our best proposal, also in

[9] http://www.msmarco.org/

[10] https://github.com/criteo/autofaiss

this case we used the extended sparse index, with $k = 1.05$ and $b = 0.7$ and then obviously the Mono-T5 model for re-ranking. With this pipeline we were able to get $nDCG@5 = 0.726$ for the relevance and $nDCG@5 = 0.699$ for the quality. Our main goal was not to get the best scores possible, therefore we did not create different models in order to obtain the best score for quality or relevance, but we found models that perform well on both.

Pipeline	**nDCG@5**
Sparse index	0.508
Dense index	0.594
Hybrid approach	0.619
Sparse with Mono-T5	0.726

Table 1: nDCG@5 scores on relevance

Pipeline	**nDCG@5**
Sparse index	0.549
Dense index	0.614
Hybrid approach	0.656
Sparse with Mono-T5	0.699

Table 2: nDCG@5 scores on quality

As far as the "stance detection" is concerned, as discussed in the section 3, we tried some different combinations of models and parameters, and ultimately chose the best configuration out of those that have been tried. However, the pipeline turns out to be very simple. Since, as we mentioned, we loaded the DistilBERT based model from Hugging Face, we also had the possibility to load an instance of the `DistilBertTokenizer` class which, automatically, applied some pre-processing techniques to the input text (such as lower-casing) and then "tokenized" the result(s), thus generating the input to feed our network(s) with. Depending on the tested networks, we performed different but similar actions before starting the training: in order to train the first network, we just had to split the data-set into train set, validation set and test set, with size proportion being, respectively, 80%, 10%, 10%; in order to train the two separate models we extracted two data-sets out of the main one: the first containing all the records, but after having made indistinguishable the stance being in favour of the first or the second object; the second containing only the records whose stance of the document with respect to the query expressed being in favour of the first or the second object: in this way, we trained the first network in distinguishing whether the stance of a document with respect to a query was favouring an object or not, and we trained the second network in detecting whether the stance of a document with respect to a query is favouring the first rather than the second object; finally, in order to train the two models and also taking into account the non-uniform distribution of the classes, we used the exact same data as before, without applying any data augmentation technique or following any other clever approach, such as random oversampling; instead, we specified another parameter while configuring the training. In particular, we gave more weights to the under-represented classes, proportionally to how much they were. In particular, suppose that the data-set contains D records, D_0 of which have been labeled as "0". The weight assigned to this class is, therefore $\frac{1}{2} \cdot \left(\frac{D_0}{D}\right)^{-1}$, where the multiplicative factor is added in order to help in "keeping the loss to a similar magnitude".[11] For all the networks we used the `SparseCategoricalCrossentropy` as loss function. We also monitored the `SparseCategoricalAccuracy` metric during training. We chose the Adam algorithm as optimizer, as it is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters" (Kingma and Ba, 2015). We tested the performances of each combination of network(s) and parameters (in a broad sense) discussed. Moreover, due to the fact that, as we mentioned, we don't have a lot of labeled data to train the model(s), we continuously compared the results with those of a random classifier, in order to keep monitoring the gain in performances with respect to this "baseline". In the tables 4, 5, 6, a summary of the performances of our networks (evaluated on the test data-set) can be found. As predicted by the aforementioned paper, we were actually able to achieve much better results when we considered two different networks working together in order to solve the task, as shown by the table 7.

[11] Tensorflow: imbalanced data

6 Discussion

For the document retrieval part, the main metric that was considered during the task, was the nDCG score. It calculates the Cumulative Gain of a set of results by summing up the total relevance of each item in the result set, the lower a relevant items is in the results, the higher the penalty, check formula 2.

$$DCG@k = \sum_{i=0}^{k} \frac{relevance_i}{log_2(i+2)} \quad (2)$$

The nDCG is a normalized value of the DCG over the ideal DCG, therefore we compute how relevant is our list of results compared to the ideal list, see formula 3.

$$nDCG@k = \frac{DCG@k}{IDCG@k} \quad (3)$$

The nDCG scores computed for each of our pipeline is the nDCG mean over 50 different queries, computed with a script provided by the challenge organizers. We limited the metric up to 5 passage texts for each query, i.e. nDCG@5, to compare our results more extensively with the other competitors. In addition, we decided also to compute the Recall@100, in order to check how many relevant documents are retrieved among the actual ones, considering the judgements given by the organizers, see table 3. The recall is an important metric in the evaluation of the retrieval stage, while it is the opposite for the nDCG, which measures the effectiveness of each ranking stage. Even though the latter is the only one mentioned by the competition we decided to consider the recall, which shows unexpected results. In fact, the dense index is the worst retrieval method, while it achieves a higher nDCG compared to the sparse index. The reason beyond this result has been already introduced in the Background section, which underlines how neural retrieval methods perform better with a limited set of documents, which is quite the opposite of the end-to-end approach we use in the dense index. As you can see, we did not compute the recall for the model that uses the re-ranking with mono-T5, this is due to the fact that the recall is not a quantitative metric of the ranking stage, but only of the retrieval, hence it is better to evaluate the re-ranking using the already discussed nDCG metric. In general, when dealing with multi-stage retrieval and ranking pipelines, which may involve subsequent operations, the recall could be used even for the re-ranking process in order to observe if the reduced corpus is still made of significant documents with respect to the query. Our baseline

Pipeline	**Recall@100**
Sparse index	0.736
Dense index	0.707
Hybrid approach	0.803

Table 3: Recall@100 scores of different pipelines

is the sparse pipeline but we can see an incremental improvement in the nDCG scores using more complex models, see tables 1 and 2. The Sparse with Mono-T5 ranking is the best one, that can reach a good score for both relevance and quality, placing us in the second position considering the relevance ranking, and in the third one for the quality.[12]

When training the neural network(s) responsible of determining the stance of a document with respect to a comparative question, we observed that using two networks under the hood with their respective objective, turned out to be more effective, compared to a single network, even before re-weighting the different classes as explained in the section Experimental setup and results. We agree on the fact that this result may be caused by the scarce amount of training data available, which does not give the networks the opportunity to generalize but, rather, to specialize in performing more particular tasks, such as the sub-problems we trained the two networks on. Still, understandably, we didn't reach outstanding scores, especially because the available data was really unbalanced in terms of the distribution of the classes: one of the main sources of error, as a matter of fact, was the almost absent prediction of the under-represented classes. However, by giving more weight during training to those classes, we were able to improve the results by quite a bit, as shown in the F1-score column of the tables 5 and 6. It goes without saying, though, that to improve the results even more, it would certainly be beneficial to gather more data to train on, since, as of January '23, the used data-set contains less than 1000 records.

7 Conclusion

Regarding the document retrieval task, we obtained good results with the hybrid model and the sparse retrieval with Mono-T5 re-ranking. Both

[12] Touché ranking

includes in some way a neural architecture, the former in the retrieval stage, while the latter only in the re-ranking. We expected a similar behaviour since Transformers are able to capture a lot of semantic information about the relationships between words in a text. However, it is remarkable how term-based approaches are still important in the first stage of the pipeline, i.e. the retrieval from a large corpus, while their importance diminish at later stages, where neural-based architecture outperforms. Furthermore, as already noticed in the literature, the importance of expansions, of both queries and documents, is negligible if considered on its own, however it becomes more effective when it is part of a whole process in the retrieval stage. This is especially the case of query expansion with RM3, which leads to the problem of query drift, thus decreasing the efficiency of the model. In the future, more detailed work could be done by combining both term-based and neural-based models, as we have done, while exploiting the fine-grained contributions of expansions. Still, it is quite notable that the importance of neural-based models is rising, especially in the ranking stage which deals with a shrunk corpus compared to the retrieval, thus it is more efficient.

For what concerns the "stance detection" task, we were surprised to discover that, even though we had less than 1000 records available to train our system built upon the two networks described in the previous section 5, the model was able to achieve good results also compared to other teams of scholars. This suggests that the chosen architecture was appropriate for the task at hand and that the techniques used to counter the non-uniform distribution of classes in the data-set were effective and successfully applied. The main drawback of the system is the required computational power, not only to train, but also to do inference, which is, currently, unavoidable with the used architectures, which also happen to be the state-of-the-art in many NLP tasks. Furthermore, the system works effectively when this step is preceded by a document retrieval procedure which gathers only the most relevant documents with respect to a specific query. In addition, it could be useful determining - with a good amount of confidence - an approximate span of text in which the answer is very likely to be found, particularly when the document is very large. Considering our chosen approach, the complete architecture which comprises both retrieval and stance detection respects these constraints. In future, it could be a good idea to try lighter models in order to speed up training and inference time, especially if, as time passes, more training data will become available.

8 List of tables

	Precision	**Recall**	**F1-score**
Macro	.31	.36	.33
Weighted	.40	.43	.40

Table 4: Statistics for the single network configuration (in the test data-set).

	Precision	**Recall**	**F1-score**
Macro	.63	.36	.35
Weighted	.54	.44	.41

Table 5: Statistics for the dual network configuration (in the test data-set).

	Precision	**Recall**	**F1-score**
Macro	.47	.55	.48
Weighted	.50	.48	.47

Table 6: Statistics for the dual network configuration with class re-weighting (in the test data-set).

	Precision	**Recall**	**F1-score**
Macro	+ 51%	+52%	+45%
Weighted	+25%	+11%	+17%

Table 7: Performance gains of the final model (dual network and class re-weighting) with respect to the baseline.

9 Links to external resources

Datasets: https://zenodo.org/record/6873567

References

Adam Berger, Rich Caruana, David Cohn, Dayne Freitag, and Vibhu Mittal. 2000. Bridging the lexical chasm: Statistical approaches to answer-finding. In *Proceedings of the 23rd Annual International ACM*

SIGIR Conference on Research and Development in Information Retrieval, SIGIR '00, page 192–199, New York, NY, USA. Association for Computing Machinery.

Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. 2018. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York. Springer.

Alexander Bondarenko, Yamen Ajjour, Valentin Dittmar, Niklas Homann, Pavel Braslavski, and Matthias Hagen. 2022a. Towards understanding and answering comparative questions. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 66–74, New York, NY, USA. Association for Computing Machinery.

Alexander Bondarenko, Maik Fröbe, Johannes Kiesel, Shahbaz Syed, Timon Gurcke, Meriem Beloucif, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2022b. Overview of touché 2022: Argument retrieval. In *Proceedings of the Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum, Bologna, Italy, September 5th - to - 8th, 2022*, volume 3180 of *CEUR Workshop Proceedings*, pages 2867–2903. CEUR-WS.org.

Yinqiong Cai, Yixing Fan, Jiafeng Guo, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2021. Semantic models for the first-stage retrieval: A comprehensive review. *CoRR*, abs/2103.04831.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: An easy-to-use python toolkit to support replicable IR research with sparse and dense representations. *CoRR*, abs/2102.10073.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. *CoRR*, abs/2003.06713.

Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. 2019. From doc2query to doctttttquery.

Rodrigo Frassetto Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *CoRR*, abs/1901.04085.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *CoRR*, abs/2101.05667.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.