Universiteit Leiden
The Netherlands

Computer Science

Zero-shot Strategies and Cosine Similarity in Selectical: a Technology Assisted Review System

Sem Kluiver

Supervisors:
Suzan Verberne
Zhaochun Ren

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

January 29, 2024

Abstract

In this paper we attempt to further develop an existing Technology Assisted Review system called Selectical, which was developed by Landscape in an attempt to speed up the paper labeling process for systematic reviews by continuously re-training a model which suggests potentially relevant papers. The objective was to achieve a very high recall of 99% or 100% of relevant papers as fast as possible. Our efforts address the cold-start problem using various zero-shot models to select the initial papers to be labeled and we attempt to find a relation between the cosine similarity between documents and their relevance. We compare various zero-shot models, and compare our additions to the original Selectical model. Although our contributions did not decrease the overall labeling load in the long run, our method did improve the speed with which the first relevant papers were found compared to the original Selectical model. We performed a thorough analysis of our results in order to identify future points of improvement.

Contents

1 Introduction

A systematic review is the process in which researchers try to assess and sometimes replicate past research in a certain field. Systematic reviews are a necessary yet very time-consuming task across many fields [OETM+15, BRKF17]. They play a large part in medical research [GG13], but also in patent search. In systematic reviewing, it is very important to review as many papers that are relevant to the research as possible. This is what makes systematic reviews a time-consuming task: finding every relevant paper requires skimming through a large collection of papers, most of which will not be relevant to the research. One can imagine why finding all relevant documents is important when taking the two mentioned examples into consideration: for medical research, it is crucial to know as much as possible about a certain condition, illness or medicine before starting research or a clinical trial [Mul94]. Having knowledge of past attempted experiments can also avoid duplicate studies and/or repeat mistakes. For patent search, reviews are also performed for a variety of reasons: for example to learn about recent inventions, or to know if a certain invention has already been patented in order to avoid patent infringement [Cla18, Lar99].

Systematic reviewing depends on effective information retrieval methods. Information retrieval for systematic reviewing consists of two steps. The first step consists of retrieving an initial set of documents, which is extracted from a much larger database. This is done to create a smaller, manageable set of papers [Mul94]. This initial set is often fetched using Boolean queries where terms in the query need or need to not be in the documents [SZK21]. The second step consists of determining which of the retrieved papers are actually relevant for the systematic review by labeling each paper, usually after reading only the title and abstract. Both the formulation of the Boolean query and the document labeling are very time-consuming tasks and are usually performed by humans.

Solutions to speed up systematic reviewing have been proposed in the information retrieval domain. Both the first and second step can be optimized: the first step to create a smaller set that needs to be labeled, and the second step to find the relevant papers faster. For the first step, the constraints on the set of initially retrieved papers could be tightened, resulting in less papers needing to be reviewed [SZK21]. For the second step, a (re-)ranking can be made that determines the order in which papers are seen by the reviewer, i.e., screening prioritization [WSKZ22]. This ranking can be more accurate than the initial ranking in the first step, since the retrieved set is smaller and therefore allows for more powerful but computationally expensive natural language processing models. Creating such (re-)ranking is a form of Technology-Assisted Review (TAR) [GC14], where an algorithm or model aids the researcher in finding relevant papers and discarding irrelevant ones during the second step. A TAR-model will suggest the researcher an order in which they should assess papers, often ranking them by estimated relevancy. The researchers then proceed to label papers, while the TAR-model can learn from this relevance feedback and will start to make better suggestions. This is called "active learning" and is a form of screening prioritization which is continuously updated. The TAR-model reduces the time required to find the desired percentage of all relevant papers. Prior work has shown that TAR-systems are beneficial for the efficiency of users in systematic reviewing [Ker05, CM09, RKO10, GC11]. Multiple TAR-models exist, such as AutoTAR [CG15] which is often still used as a baseline for other TAR-models. AutoTAR and many other TAR-models do not use modern transformer models for better natural language processing. Most models do not retain knowledge between labeling sessions about different topics and are prone to the cold-start problem.

In this thesis, we will focus on speeding up the labeling process of papers for a TAR-system called Selectical.[1] This model was developed by Landscape, a company based in Leiden which focuses on AI solutions.[2] Selectical is currently aimed at speeding up the paper selection process in the medical domain and fulfills two tasks [SZK+17]: it performs both screening prioritization as well as deciding the stopping point at which the reviewer should stop reviewing. It is a relatively recent TAR-system which uses embeddings generated by transformer models as a feature. However, it has, like other models, no way of retaining knowledge of past labeling sessions and may be prone to the cold-start problem. We will discuss our attempts at creating better rankings, as well as our contributions to Selectical in an attempt to speed up the paper selection process. We have taken multiple approaches for this task, and will describe the results and findings of each of these methods. To summarize the goal of this thesis in scientific metrics, we try to achieve the highest possible precision at a recall of (nearly) 100%. In other words, we try to have no false negatives, while getting as few false positives as possible. Our research questions are therefore:

1. **"How can we achieve the best possible precision with a recall of (nearly) 100% using various ranking strategies, either stand-alone or in tandem with Selectical?"**
2. **"How do various models in a stand-alone setup compare in terms of performance?"**
3. **"What is the best approach to integrating or adding new features or components to Selectical using the knowledge gained by using stand-alone models?"**

We make the following contributions in this thesis:

- We compared the performance of various types of ranking models as re-rankers for an initial set of documents retrieved with a Boolean query.
- We attempted to alleviate the cold-start problem of Selectical, where the active learner suggesting the next document to label has not seen any relevant documents at the start of a labeling session.
- We implemented a novel component which can be added to the active learner of Selectical. It is trained on the relation between document relevancy, and the similarity between the current document to be labeled and all documents that were labeled so far.

2 Background of the used models

This section describes the different models and methods that are used in this thesis. We explain the basic architectural aspects of each model, while explaining why those approaches should intuitively work for our problem. Considering that no labels of the documents are available in real-world systematic reviewing tasks, most models are unsupervised. These models assess document relevancy by comparing the words used in the query and document. More complex models make use of document embeddings and calculate similarity scores. Some models are pre-trained to learn to rank [Liu09].

[1] https://wearelandscape.nl/whitepapers/selectical
[2] https://wearelandscape.nl/

2.1 Transformer models and BERT

The development of transformer models proved to deliver a large increase in performance in natural language processing tasks. One of the main aspects making transformer models so powerful are the self-attention modules [CDL16], which relates each word in the input to each other word (and itself) in the input. This enables transformer models to understand context much better than previous methods. One of the state-of-the-art natural language processing tranformers is BERT [DCLT19]. BERT can calculate embeddings for any particular text, with a maximum length. First, BERT tokenizes words, which helps in understanding various conjugations and different forms of the same word. For example, "walking" is split in "walk" and "ing", such that BERT understands that the verb is about the act of "walking", which is in essence the important part: the conjugation is not relevant. BERT also has various tokens that can be used for indicating aspects of texts. The `[CLS]` token indicates that BERT should perform (sentence) classification. The `[SEP]` token is the separator token, which is placed between sentences or documents to delineate them. The `[UNK]` token replaces tokens that are not in the model's vocabulary. After the tokenization step, BERT computes the embedding for the text. These embeddings contain meaningful information about the text. Originally, BERT was trained in two tasks on the Toronto BookCorpus and the entirety of the English Wikipedia. The first task is predicting what word should be in a sentence where a word was masked. The other task is predicting, given a certain sentence, whether another sentence would be the next sentence in the text or not. Both of these training tasks make BERT learn about context. The BERT model can later be fine-tuned on smaller, more specific datasets. A huge number of such fine-tuned BERT models exist, such as BioBERT for biomedical texts, and SciBERT for scientific texts. BERT models and their variations can be used for a multitude of tasks, including the tasks it was trained on (next-sentence prediction and masked word prediction) as well as classification, named-entity recognition, and more. During these tasks a BERT-model can be fine-tuned further. For each task a (slightly) different setup in terms of inputs and outputs is used. Figure 1 shows the architecture of a BERT model, set up to take two sentences as an input and perform a sentence pair classification task (such as next-sentence prediction).

Multiple variations of the BERT model have been made including BERT-based models for document ranking. Examples include monoBERT and duoBERT [NYCL19]. Both architectures aim to rank papers based on relevancy to the query. The main difference is that monoBERT is a point-wise model, and duoBERT is a pair-wise model; i.e., monoBERT takes the query and a single document as an input, and duoBERT takes the query and two documents as an input, with the constraint that one document needs to be relevant to the query, while the other needs to be irrelevant to the query. Both models subsequently feed the output vector of the `[CLS]` token into a single-layer neural network. The authors propose a retrieval pipeline in three parts, namely, starting with BM25, of which the top-k results are reranked by monoBERT, of which in turn the top-k results are reranked again by duoBERT. Both architectures were trained on MSMARCO [NRS+17] and TREC CAR [DVRC17].

Since the great success of BERT, many more transformer models were made including the T5 transformer [RSR+20], which aims to combine multiple tasks in a single transformer. These tasks include translation, summarization and sentence similarity scoring. Each task is formulated as a text-to-text task, which allowed the authors to use the same model, loss function, hyperparameters, etc. for each task. Using the T5 transformer, a monoT5 [NJL20] model was also made for ranking documents with queries, which works in a similar way to monoBERT.

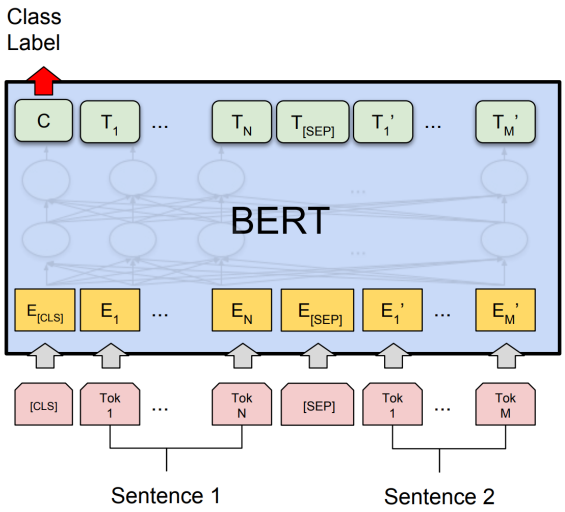


Figure 1: The architecture of BERT, in this case taking two sentences as input and outputting a classification. This could be next-sentence prediction, but also sentence similarity, or some other task. Note the `[CLS]` as the first input, and the `[SEP]` token between the two sentences. Image taken from [DCLT19].

2.2 Active Learning

Considering that each systematic review is different, it is hard to train a general model that can be used for each review. Technology Assisted Review systems therefore often have self-learning components that are trained on the relevance feedback of the reviewer labeling the papers. The next paper the reviewer will see is then decided by the learning system. These self-learning systems using human feedback are called Active Learners. Generally, there are two types of Active Learning systems: Simple and Continuous Active Learners [CG14]. Simple Active Learners (SAL) show the reviewer the paper the model is the most uncertain about. Intuitively, when this paper is labeled, the learning system has the largest knowledge gain of what papers are considered relevant and which ones are not. The other type of active learners, called Continuous Active Learners (CAL) show the reviewer the paper it thinks is the most relevant first. Intuitively, the reviewing process will speed up the most, since the reviewer will label the relevant papers faster. Active learning models are somewhat supervised, since they learn from human feedback during labeling time.

2.3 Existing model architecture: Selectical

Selectical is a Continuous Active Learning model which takes multiple textual features as its input. It was specifically built to perform relevance assessments on the given dataset we will describe in Section 4. The data for which we try to speed up the review process is split up in multiple subsets of papers that were retrieved using a certain Boolean query. Each subset represents a topic. The aim is to find the relevant papers as soon as possible for each subset. Each paper has a title, abstract, keywords, authors and relevance label (given by humans). Of course, in real-world scenarios, this last label is not available. We only have access to the Boolean query, and not to the research question formulated in natural language. These features are all individually preprocessed in a model or neural network which is not trained. The output of each preprocessing networks is fed

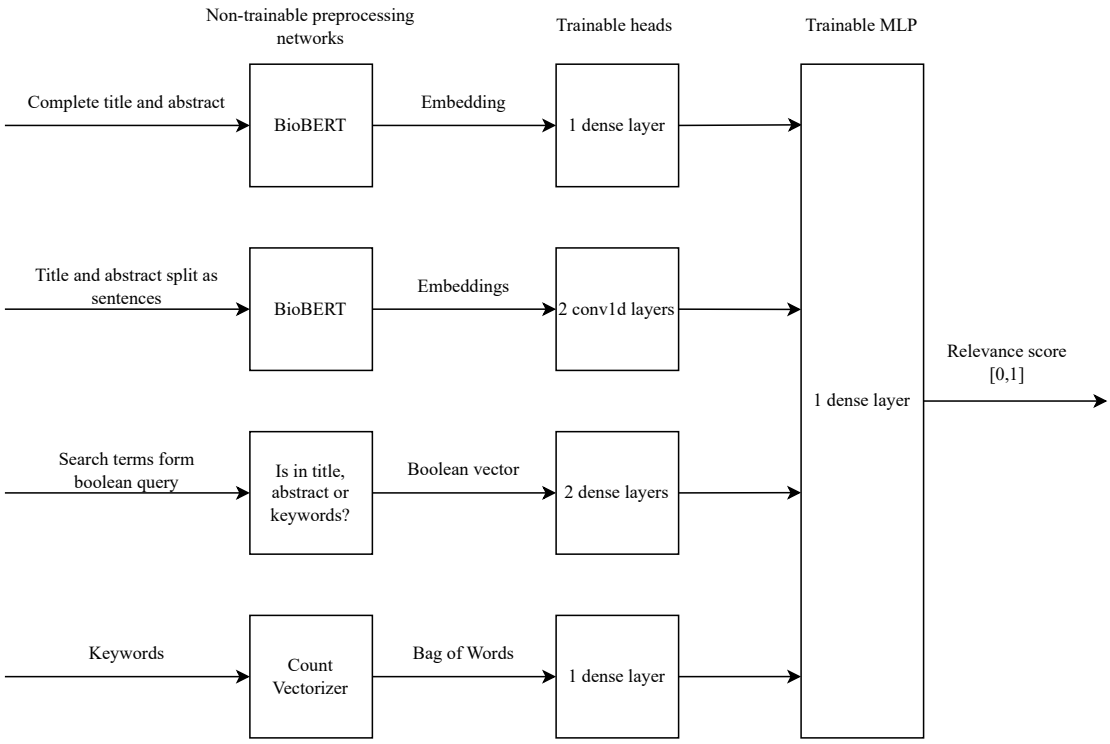


Figure 2: Selectical in its current state has four heads, each taking different features from the documents. Only the heads and small MLP are trainable to ensure active learning is responsive.

into of of four different heads, which is a trainable model. These four features used by the heads are:

1. The embeddings of the title concatenated with the abstract, computed by a pre-trained transformer model. Selectical uses BioBERT-v1.1 [LYK+19].

2. Each individual sentence of the title and abstract embedded on its own by the same transformer model. This was done for redundancy. These embeddings are then fed into a convolution layer.

3. A Boolean vector that denotes whether or not the terms in the Boolean search query occur in the title and abstract.

4. A bag-of-words vector containing the keywords of the paper.

The output of each head is fed to one multi-layer perceptron (MLP) which makes the final prediction and gives an estimate whether a document is relevant or not. Scores are in the range of [0,1], and higher means higher estimated relevance. This MLP is the active learner of Selectical. During the labeling process the reviewer will label the document with the highest score assigned by the MLP. Every time a new paper has been labeled, this neural network is reinitialized and retrained on all papers that were labeled so far. Papers that were labeled are taken out of consideration and are not scored again. Note that none of the individual components, including the transformer model, is trained at any point in the reviewing pipeline; only the MLP and heads at the end of

the pipeline are retrained. This was decided such that the model would be computationally cheap to re-train during labeling, which is a requirement for active learning models. An overview of the model architecture can be seen in Figure 2.

Selectical has a few shortcomings. For example, the model has to be trained from scratch each time a new topic is labeled. This results in the model suffering from the cold-start problem. As long as the model has not encountered one of the relevant papers, it will not have any example of a relevant paper to train on. Selectical therefore has the option to be initialized using seed papers, however these are not always available in real-world scenarios. Additionally, no knowledge is kept between topics, even though papers across topics might share features attesting relevance. Selectical also only uses the search query in a very basic way, namely, by creating a Boolean vector denoting whether the terms in the query occur in the paper. This is a relatively simple representation of the query, and more features could potentially be extracted by encoding the query in a more meaningful way. Intuitively, the query describes what the researchers are looking for, so it is important to take it into account as best as possible during the ranking of papers.

2.4 Transfer learning

Intuitively, papers that are relevant for a certain topic may share textual or structural features with papers that are relevant for other topics. An example of possible knowledge that may be useful for multiple studies that could be carried over with with transfer learning is the type of publication, type of study, whether it was carried out on humans or animals, age groups, ethnic groups, countries and regions, and other similar study properties. Structural features may also give a general sense whether or not a paper is relevant or not. For example, recent papers are usually more relevant than older papers, and papers with a title containing a question are usually not relevant.[3] Transfer learning can alleviate the cold-start problem by giving a model an initial edge, rather than it having to learn from scratch.

2.5 Lexical Rankers

Simple lexical models that determine paper relevance by using term overlap have existed for a long time, and often still provide a strong baseline. These lexical rankers are all unsupervised and attempt to rank documents based on word occurrence in query and document. Examples include TF-IDF and BM25. Even though lexical rankers are generally older models, BM25 is up to the present time used in initial retrieval steps since it is so cheap to compute while still giving relatively good initial rankings. Lexical rankers are exact-match methods. Therefore, performance may increase when words are reduced to a generalized form. Preprocessing steps therefore often include "stemming" or "lemmatization". When stemming words, they are changed to their stem, the common part of all inflections: "change" and "changing" become "chang". Using lemmatization, words are converted to their lemma: in the stemming examples, all words would be converted to "change".

The rankings produced by computationally inexpensive lexical rankers can be re-ranked by some more computationally expensive model. However, since these rankers are a cheap way to

[3]This is described by Betteridge's law of headlines, which states that the question asked in titles of articles can usually be answered with "no". https://web.archive.org/web/20090226202006/http://www.technovia.co.uk/2009/02/techcrunch-irresponsible-journalism.html Date accessed: 21-12-2023.

produce a meaningful initial ranking, we could also use them to determine the first documents the reviewer should label using our TAR-model. The set retrieved using a Boolean query is unordered and the active learner therefore at first randomly proposes documents to the reviewer. Intuitively, a lexical ranker should find some relevant documents faster than random selection, and therefore the active learner should be able to learn a relevant document's properties faster. The active learner could then take over the decision making after a number of documents have been labeled. We now explain the intuition behind some lexical rankers.

TF-IDF stands for "term frequency – inverse document frequency". It is based on two intuitions: if a term we are looking for with a query often occurs in a document, that document is likely relevant. However, a term occurring in a large amount of documents should not be weighted as much as terms that are quite rare and only occur in some documents in the collection. This is where the "inverse document frequency" comes into play: the more documents contain a term, the fewer it weighs. Intuitively, words occurring in all texts do not carry much or any information, so they should not be weighted as much as very specific words that we are looking for. These often-occurring words that do not contain a lot of information are called stopwords. Examples include "a", "the", "of", "from", etc. These are sometimes removed from both documents and query before scoring in order to increase information density. Generally, TF-IDF is calculated with Equation 1.

$$\text{TF-IDF}(t, D) = \text{tf}(t, D) \cdot \text{IDF}(t) \tag{1}$$

Here, $\text{tf}(t, D)$ is the number of times term t occurs in document D. $\text{IDF}(t)$ is often calculated with Equation 2.

$$\text{IDF}(t) = \log\left(\frac{N}{\text{df}(t)}\right) + 1 \tag{2}$$

In this equation, N is the total number of documents in the collection, and $\text{df}(t)$ is the document frequency of term t, i.e., the number of documents in the collection that contain term t. The $+1$ term exists to ensure the score is non-zero.

BM25 is in essence quite similar to TF-IDF, but has a slightly more complex calculation which takes a few extra aspects into account in its calculation. Various implementations of BM25 exist. Some, like BM25F, can take different fields into account (such as title, abstract, keywords, MeSH terms[4], etc.), others, like BM25+, were specifically designed to not penalize longer documents too much. The default computation method for BM25 can be seen in Equation 3.

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avg dl}}\right)} \tag{3}$$

Here, $\text{score}(D, Q)$ denotes the score of document D with query Q. The equation sums over all keywords q_i in query Q (with length n). $f(q_i, D)$ is the frequency of term q_i occurring in document D. k_1 is a parameter, where usually $k_1 \in [1.2, 2.0]$. b is another parameter, usually $b = 0.75$. $|D|$ is the length of document D, and avg dl is the average document length of all documents in the collection. $\text{IDF}(q_i)$ is the inverse document frequency of keyword q_i. The inverse document frequency ensures that words that occur in a large number of documents weigh less when scoring. The inverse document frequency used in BM25 is calculated using Equation 4.

[4]Medical Subject Headings, i.e, medical vocabulary used for indexing articles.

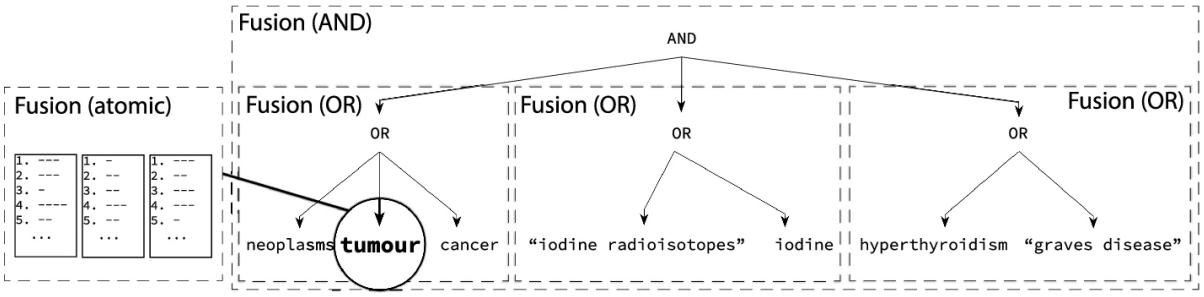


Figure 3: RankFusion CLF merges rankings. First, atomic rankings are made and fused. Then, based on OR and AND operators, these rankings are merged in turn. Image taken from [SZK20].

$$\text{IDF}(q_i) = ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right) \tag{4}$$

Here, N is the number of documents in the collection. $n(q_i)$ is the number of documents that contain term q_i. The $+0.5$ terms ensure that no division by zero is made, and that the fraction cannot amount to 0. The $+1$ term after the fraction ensures that the natural logarithm does not result in a negative outcome, which would result in a negative score, since the IDF is multiplied with another term in Equation 3.

2.6 RankFusion CLF

Since BERT-models and lexical rankers are not specifically designed for encoding and understanding Boolean queries, we searched for a method that uses Boolean queries to rank documents. One of such rankers is called RankFusion CLF [SZK20]. This method ranks documents using any or multiple (light-weight) ranking method, such as the lexical rankers described in Section 2.5: it can use BM25, TF-IDF, etc. for each subclause or atomic term (i.e., each separate word occurring in the query) and then merges these rankings. Intuitively, papers that receive high scores from multiple rankers are likely to be relevant. As RankFusion CLF is a method that merges unsupervised rankings, it is itself unsupervised as well. Each ranking is merged using one of two methods, depending on the OR and AND operations connecting the subclauses. NOT operations are not supported. The bottom-up ranking process can be seen in Figure 3.

The rankings for subclauses are merged based on two formulas: for merging the atomic rankings and rankings connected by OR operators, CombMNZ [FS94] is used. For merging rankings connected by AND operators, CombSUM [FS94] is used. Formally, a set of rankings $R = r_1, r_2, ..., r_k$ exists for each subclause. For each atomic clause, one or more rankers can be used. Each ranking is an ordered list of documents, where each document is assigned a score by that ranking model: $r_n = d_1, d_2, ...d_k$ where each document-ranker pair has a score: $s(d_i, r_j)$ for document d_i in ranked list r_j. These scores are merged by the aforementioned formulas, formally written as seen in Equation 5, where T denotes the type of clause to be merged.

$$f_{CLF} = \begin{cases} \sum_{r_j \in R} s(d, r_j) & \text{if } T = \texttt{AND} \\ |d \in R| \cdot \sum_{r_j \in R} s(d, r_j) & \text{if } T = \texttt{OR/Atomic} \end{cases} \tag{5}$$

In the original paper, the authors use an ensemble of (simple) ranking models to rank each atomic clause. These rankers are: IDF, TF-IDF, BM25, InL2 of Divergence from Randomness, term position, publication date, document length, text score, and PubMed weighting. To further elaborate: term position is scored by the relative position of the word in a text. Publication date is scored based on how recent the paper was published. Document length scores are based on the length of the document. Text score determines in what fields a word occurs in the document, and ranks documents higher when the word is in both the title and document body. The PubMed weighting works by ranking in three stages: documents are retrieved by the Boolean query, then ranked using BM25, and lastly the top-n documents are re-ranked by LambdaMART, trained on click data with metadata as features. This metadata includes document length, publication date, and past usage. No stemming, lemmatization or stopword removal is performed. Before merging, each score of each clause is mapped to $[0, 1]$ because all rankers provide a very wide range of scores. The authors ran their experiments using QueryLab [SLZ18].

2.7 Transformer-based models

Since transformer models like BERT provide such powerful textual embeddings and has been applied in many natural language tasks, we implement some BERT-based ranking models specifically designed for information retrieval and/or textual similarity tasks.

2.7.1 SentenceBERT: bi-encoders and cross-encoders

SentenceBERT [RG19] is a variation of the original BERT model which was developed to speed up the ability of comparing textual similarity. Until SentenceBERT, similarity between texts was calculated with BERT-based models by concatenating the two sentences or texts into a single input, after which BERT could output a prediction denoting whether the two sentences were related or not. This model architecture falls under the category of "cross-encoders". One can imagine that this is very computationally expensive, since each combination of sentences (or texts and queries) needs to be encoded in order to compare all textual pairs.

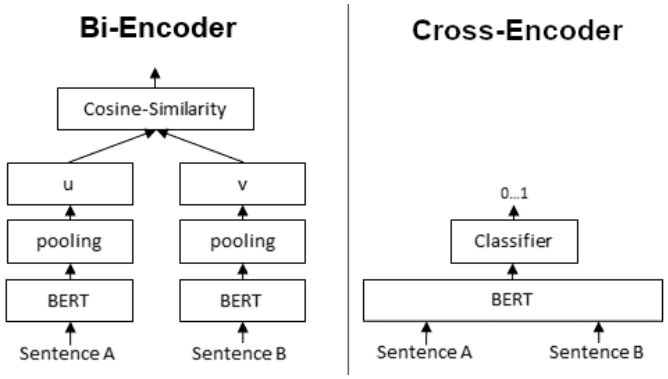


Figure 4: The architecture difference between bi-encoders and cross-encoders.[5]

[5]Image taken from https://www.sbert.net/examples/applications/cross-encoder/README.html. Date accessed: 15-05-2023.

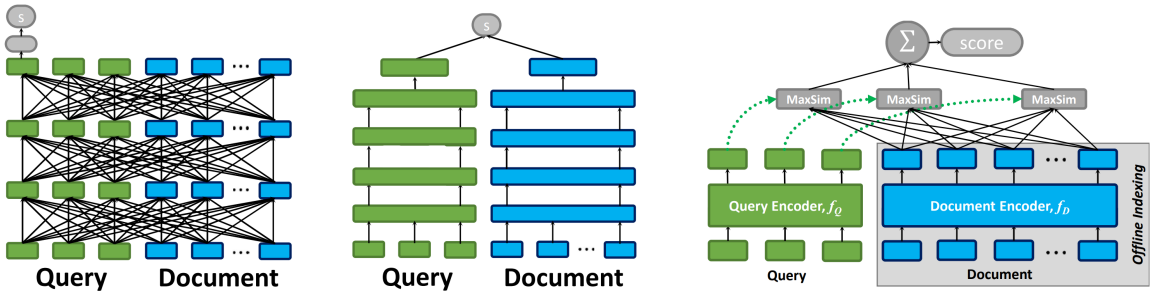
SentenceBERT, which falls under the category of "bi-encoders", solves this problem by being able to encode each sentence or text separately, of which the embeddings can be compared afterwards using cosine similarity. The difference in architecture can be seen in Figure 4. SentenceBERT was made by fine-tuning a BERT model and building Siamese and triplet networks [SKP15]. Siamese networks are focused on having two models with the same weights produce the same output, given a different input. Triplet loss is made specifically for comparing a reference input with a positive and negative input, and maximizing the output similarity between the reference and positive input, while minimizing the similarity between the reference and negative output. Intuitively, SentenceBERT is in this way trained to, given two different input texts with similar meaning, output a similar embedding, while two texts with different meanings result in two dissimilar embeddings. The similarity of two embeddings can be calculated using the cosine similarity. SentenceBERT can use any standard transformer in its training process and therefore has the same possible input length as the transformer model it was based on. Like standard transformers, SentenceBERT can also be fine-tuned on specific tasks.

For each text, its embedding needs to be calculated only once, which is the computationally expensive part. This way, SentenceBERT can be used to compare the meaning of texts while being much less computationally expensive than BERT or cross-encoders, where encoding every combination of texts is necessary to determine similarity. Since SentenceBERT is able to determine whether two texts have similar meaning, it can also be used to determine the similarity between a query and document. Theoretically, a document that is similar to the query should be relevant to the query. Likewise, a document that is similar to a relevant document has a higher probability to also be relevant.

When computational limits are a concern, bi-encoders are generally the preferred choice. However, since there is no direct interaction between different sentences or, in our case, the queries and documents, the performance of bi-encoders may be worse than the performance of cross-encoders. Therefore, if computational time or power is not a limiting factor, cross-encoders are the better choice, since BERT's interaction can then work between sentences or query and document. In information retrieval pipelines, the sensible choice would be to retrieve an initial set of documents using bi-encoders, and then re-ranking this set using cross-encoders. This limits the computational time needed, while still giving the best performance on ranking papers that are already of interest. While this thesis focuses on the second step of re-ranking a retrieved set of papers, these sets are still quite large, which may result in long computation times using cross-encoders.

2.7.2 ColBERT

One disadvantage of SentenceBERT is that there is no direct interaction between the query and document since they are encoded separately from each other. BERT-based models are very good at understanding context and relation between different parts of texts, so some performance will be lost due to the lack of interaction between the two separately encoded texts. ColBERT [KZ20] aims to solve this problem by using a relatively cheap late interaction step between a query and document. Just like SentenceBERT, ColBERT encodes the queries and documents separately. The main difference is that SentenceBERT calculates a single embedding for each input text and thus only a single similarity for each pair of inputs, while ColBERT creates embeddings for each query and document token. For each query token the maximum cosine similarity to all document tokens is computed using a MaxSim operator. All these similarities are then summed,



(a) BERT cross-encoder all-to-all interaction. Note that each query token has interaction with each document token, which is computationally expensive.

(b) SentenceBERT bi-encoder encoding-based similarity. Note that there is no interaction between query and document tokens. A similarity score is computed afterwards, based on the embeddings.

(c) ColBERT with late interaction. There is no interaction during encoding time, but a multitude of similarities is calculated between each query and document token after encoding. This is not much more computationally expensive than SentenceBERT while containing more relations between query and document tokens.

Figure 5: Comparison of model architecture between BERT-based models. Images taken from [KZ20].

and documents are ranked based on this number. This is only a slightly more complex computation than using bi-encoders and still much cheaper than using cross-encoders. ColBERT theoretically retains more information due to taking every token into account separately. Figure 5 shows the difference in architecture between BERT (cross-encoders), SentenceBERT (bi-encoders), and ColBERT, in Subfigures 5a, 5b and 5c, respectively.

3 Related work

In this section, we will discuss previous work in the field of speeding up the document selection process for systematic reviews using various methods. We describe various techniques for Technology Assisted Review systems, and other relevant work in speeding up systematic reviews. We also discuss some general work in the field of query-document relevancy.

3.1 Autonomous Tar

One of the most well-known Technology Assisted Review (TAR) algorithms that implements Continuous Active Learning is called Autonomous TAR or AutoTAR [CG15]. This method uses an SVM classifier and uses sampling to train the classifier. Currently, AutoTAR is claimed to be one of the best-performing TAR-algorithms. It works as follows:

1. A relevant seed (intial) document is given, or, if not available, created from the topic description. The training set consists of this document only.

2. The batch size B is set to 1.

3. 100 randomly chosen documents from the collection are labeled as "not relevant" and are added to the training set.

4. The SVM classifier is trained on the train set.

5. The random documents are removed from the train set.

6. Now, after training, the classifier selects the highest B ranking documents for review by a person, who labels them as "relevant" or "not relevant".

7. These documents are added to the training set.

8. The batch size B is increased by $\lceil \frac{B}{10} \rceil$.

9. Steps 3 through 8 are repeated until the desired number of relevant documents have been reviewed.

In another paper, AutoTAR is expanded upon by giving setence-level relevance feedback [ZCGS20]. While this does not increase the recall or precision in itself, it does, as the authors describe it, reduce the effort needed to review papers. They measure the effort in two ways: the number of papers that had to be reviewed to achieve a certain recall, as well as the number of sentences that the reviewer had to read. With this method reviewers can be presented with just a single sentence estimated to be important of the papers that are to be classified. It obviously takes less effort and time to read a single sentence compared to reading an entire abstract. There are multiple settings in this algorithm, for example, the classifier can be trained on the document containing the most relevant sentence, or on the most relevant sentence in the most relevant document. The training pool can also consist of 100 random documents, or 100 random sentences. The best-performing configurations were the original AutoTAR (present most relevant document, train on documents, show best document) and the variant that presented only the most relevant sentence (present most relevant sentence, train on documents, show best sentence from best document). The results show only a slight decrease in recall when presenting the reviewer only with sentences compared to whole documents, while the measured effort is much lower.

3.2 Seed-driven document ranking

Quite often when using a TAR-system with Active Learning, a number of (most often relevant) seed documents may be given to the model. This way the model has an initial set to train on to alleviate the cold-start problem. The model is then able to give better suggestions from the start. One implementation is Seed-Driven Document Ranking (SDR) [LS18]. The authors experimented with various ways to represent documents, as well as how similarity between seed documents and the to be labeled collection is calculated. For example, rather than using the standard bag of words representation, they also implemented a bag of clinical terms representation. This can be used in all classic ranking methods, such as BM25. The bag of clinical terms representation, however, did not consistently perform better than the bag of words models.

The authors implemented a function to rank based on the cosine similarity between TF-IDF vectors among others. They also experimented with ranking based on word embeddings similarity,

that were encoded using word2vec [MCCD13], which they called Average Embedding Similarity (AES). Ranking based on both SDR and AES yielded the best results, outperforming classic ranking methods such as BM25. These results were re-confirmed by [WSMZ22]. The authors of this paper also experimented with using multiple seed documents, which they called multi-SDR. The authors concatenate all seed documents and create a single embedding. This outperformed the standard SDR. One can conclude that by having one or a few relevant seed documents to start from, identifying the other relevant documents should be easier. The authors of the reproducibility study note in their future work that using BERT embeddings may yield better performance than using the word2vec embeddings.

3.3 Zero-shot classification using pre-trained models

Aside from active learner strategies, technology assisted review systems may also implement models that have been pre-trained on large amounts of data in an attempt to generalize the labeling process between studies. Models that have been pre-trained may be able to discern relevant papers from nonrelevant ones more easily. An example of a relatively simple ranking model for biomedical texts is a cascade model consisting of four models [AD20]. Using the MEDLINE dataset, papers were rated on four criteria, in this order:

1. The format of the article. This is either an original study, case report, review, etc.
2. Whether the study concerns human healthcare or not
3. The purpose of the study. Diognosis, prognosis, treatment, etc.
4. The rigor of the study: whether it was executed correctly, depending on the purpose.

A cascade learner was then trained on the labeled papers. For each of the tasks mentioned above, a separate SciBERT-model was trained. Each SciBERT-model was only trained on the papers that received a positive label by all previous models, i.e. the fourth model was only trained on papers that received a positive label by the models that classified the previous three criteria, and the third model was only trained on the papers that received a positive label by the previous two models, etc. This cascade learner received a much higher precision than other models they compared to. The authors also experimented with another model: the Individual Task Learner (ITL). This model consists of just a single SciBERT model that feeds its output to a neural network, which classifies whether the document meets all four criteria or not. While this model has a lower precision, its recall was much higher.

3.4 Transformer models for ranking

Generally speaking, BERT and other transformer models are able to generalize well in natural language tasks, including systematic reviews. Since the release of BERT models some attempts at TAR have also been made with BERT-based models. One example is CALBERT [SC22], which used the AutoTAR model described in Section 3.1 to create an initial ranking. The top k documents were re-ranked using monoBERT, a cross-encoder. However, the recall achieved was not very high, and the authors concluded that no obvious advantages for transformer-based models were found in high-recall tasks.

3.5 Boolean query generation

The first step in information retrieval often involves Boolean queries, which are used to obtain an initial set of papers that need to be checked for relevance. The Boolean query therefore has a large impact on what papers will be reviewed, and which are excluded. In addition, this initial retrieval set can vastly change in size depending on what query is used. A fine balance needs to be found between a broad query that includes as many (relevant) papers as possible, and a very specific query that excludes as many (nonrelevant) papers as possible. We want to review as many papers as necessary to find all relevant documents, however, the initial set should have the smallest size possible to save time in reviewing papers. Formulating a Boolean query is therefore a time-consuming and difficult task, especially since it may involve terminology of specific research areas the information retriever is not familiar with. Using relevant seed documents, a Boolean query can be automatically generated in two ways, which have been compared [SZK21]: Using the conceptual method [LMG08], concepts and the research questions of the seed documents are identified. Synonyms for terms are added as well. When using the objective method [HWKS12], some seed documents are used to identify terms, and statistics are used to determine which terms should be included and which ones should not. The authors concluded in their comparison that the objective method is more effective, however, the conceptual method is easier to manually refine after automatic generation. If no refinements are made, both automatic methods performed worse than the original queries formed by humans. However, if refinements are made, a higher precision and nearly the same recall as using queries made by humans can be achieved.

Boolean queries retrieve documents by using exact-matching. This returns a set to the retriever, however, it does not provide a ranking. Some attempts have been made to produce rankings based on Boolean queries, such as RankFusion CLF [SZK20], as was discussed in Section 2.6.

4 Dataset

The data that is available to us consists of 39 medical topics, each with a number of relevant papers and a (much larger) number of irrelevant papers. On average, a little less than 8% of the papers per topic are relevant. The average data distribution can be seen in Figure 6. The full distribution for each separate topic can be found in Figure 12 in Appendix A.

For each paper the title, abstract, and keywords are available. Each paper has been labeled by medical domain experts. Both a first and second judgement are available: The first judgement was made after only the title and abstract were read. If a paper was deemed relevant, the full text was retrieved and read. The second judgement was made when the whole paper had been read, and is only filled in if it is different from the first judgement. We leave out the second judgement in this research, since we are interested in speeding up the relevance assessment process, i.e., the initial judgement made when only the title and abstract are available. For both the first and second judgement, three relevance labels are available: include, doubt, and exclude. We change the doubt label to include, since reviewers will want to read the whole paper if the initial judgement is not conclusive on whether or not the paper is relevant. After all, we want to achieve a recall as high as possible, and only want to exclude papers when we are absolutely certain they are not relevant. Note that, since the second relevance judgement may be different from the first, our data may contain some wrongly labeled relevance judgements which introduce noise. This could hamper model performance.

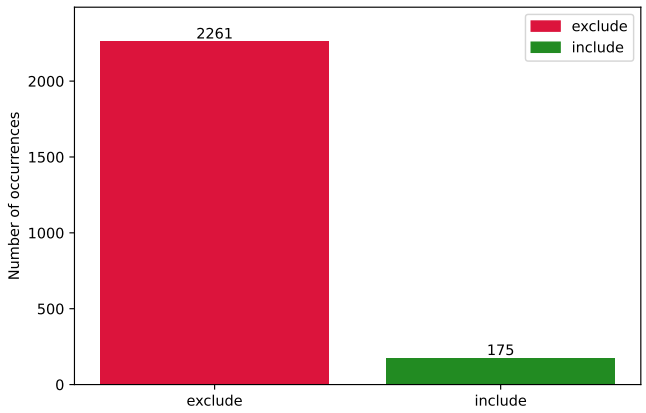


Figure 6: Average distribution of relevant and nonrelevant papers. Papers labeled as "include" and "doubt" are both visualised as "include".

Each topic also includes a file with one or more Boolean queries that have been used to retrieve the initial set of papers for that particular topic. These Boolean queries also contain specifiers that determine in what field that word should be looked for. These fields include MeSH (Medical Subject Headings), the abstract, title, keywords, and many more. Some topics have multiple Boolean queries that each look for a specific subject for that topic. The query files do not have a standardized format. Some files include a descriptive title above a Boolean query to denote what that particular query aims to retrieve (or exclude). Some files have Boolean operators between the multiple queries, while the queries of other files are only separated by newlines. One example query for the achondroplasia topic has no such descriptors but has AND operators between subqueries:

```
("achondroplasia"[MeSH Terms] OR "achondroplasia"[All Fields]) OR
↪  achondroplastic[All Fields] OR ‘‘skeletal dysplasia’’[all fields]
AND
"Prevalence"[Mesh] OR prevalen*[tiab] OR "epidemiology"[Mesh] OR
↪  "epidemiology"[subheading] OR epidemiol*[tiab] OR burden[tiab] OR
↪  "Incidence"[Mesh] OR inciden*[tiab]
```

The following query for the ADPKD topic has descriptors for the subqueries, but no operators between subqueries:

```
#1. ADPKD
"Polycystic Kidney, Autosomal Dominant"[Mesh] OR "Autosomal dominant polycystic
↪  kidney disease"[tiab] OR ADPKD[tiab]

#2. Glomerular Filtration Rate
"Glomerular Filtration Rate"[Mesh] OR eGFR[tw] OR "estimated glomerular
↪  filtration rate"[tw] OR "renal function"[tw]
```

Topics of the studies include: achondroplasia, pancreatitis, asthma, the effect of alcohol on the brain, Hepatitis B and C, Herpes, meningococcal diseases, vaccinations, and nutritional advice for pregnant women. All topics can be identified in Figure 12 in Appendix A.

4.1 Cosine similarity between papers

To investigate if cosine similarity between documents could help in the prediction of document relevance, we calculated a distance matrix of the cosine similarity between each document. We encode each document using the `all-mpnet-base-v2` bi-encoder. We then calculate the cosine similarity between each document pair. This way, we compute the similarity between all relevant-relevant document pairs, and between all relevant-nonrelevant document pairs. Intuitively, relevant documents should, on average, have a higher similarity to other relevant documents compared to nonrelevant documents. This is confirmed by Figure 7.

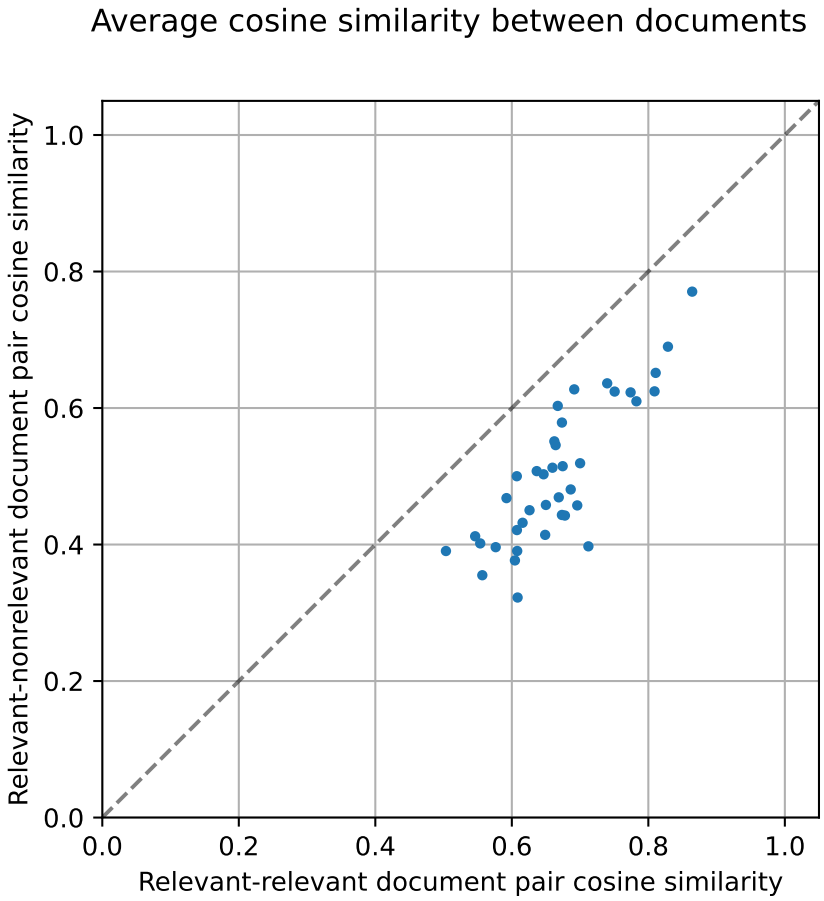


Figure 7: The average cosine similarity between papers. For each topic, we calculate the cosine similarity between each pair of papers. The average similarity was taken for both all relevant-relevant pairs, and relevant-nonrelevant pairs. These have been plotted against each other. We see that relevant-relevant document pairs indeed have a slightly higher similarity to each other, compared to relevant-nonrelevant document pairs.

To further investigate this relation, we plot the distribution of all cosine similarities for both relevant-relevant and relevant-nonrelevant document pairs. For completeness, we also show the similarity between all nonrelevant-nonrelevant document pairs, although we mostly want to discern between relevant and nonrelevant documents. However, it still provides useful information, such as whether nonrelevant documents are somewhat similar to each other. If this is the case, this

could give leads to discarding documents which are similar tot nonrelevant ones. We choose to bin the cosine similarities together in one of 20 equally-sized bins to produce a smoother curve. For each relation (relevant-relevant, relevant-nonrelevant, nonrelevant-nonrelevant) we take all similarities and show the relative proportion of similarities for that relation. We choose to show the distributions of four topics with dissimilar distributions. These are shown in Figure 8. The plot containing the distributions for all topics can be found in Figure 13 in Appendix A.

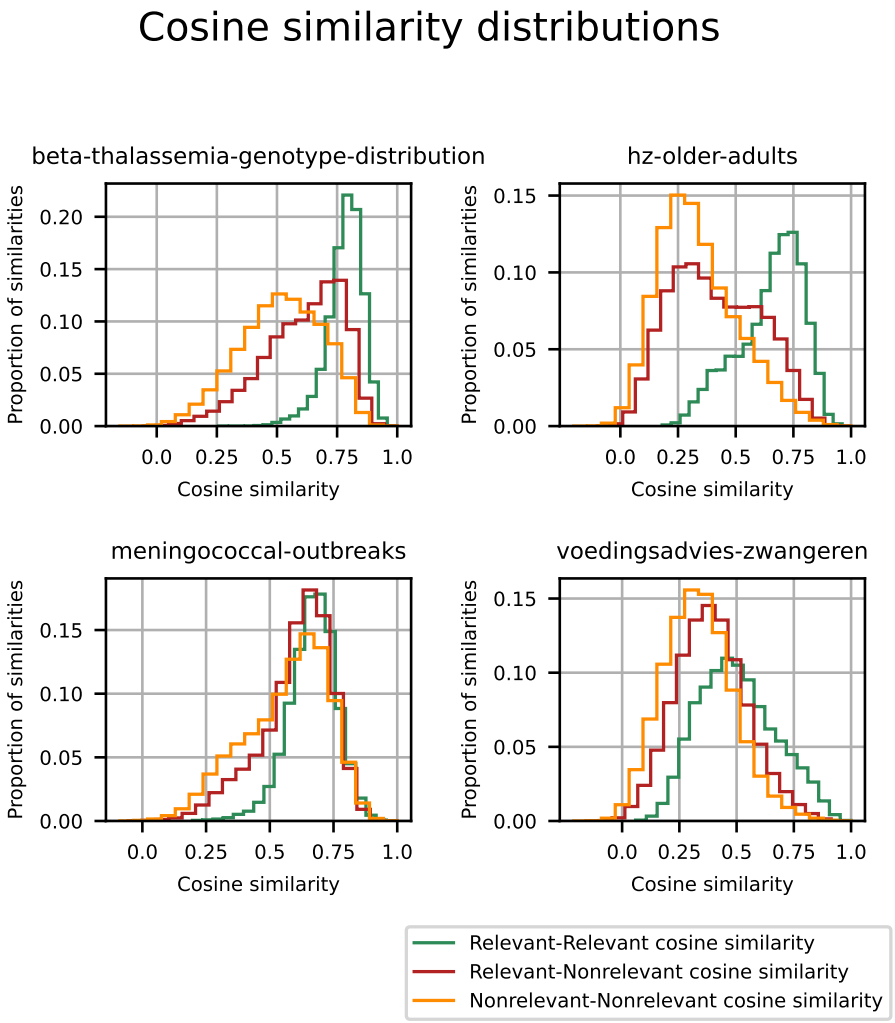


Figure 8: Cosine similarity distribution of all pairs of papers for four topics. For these topics, we calculate the cosine similarity between each pair of papers and show the distribution of similarities for each possible relation (relevant-relevant, relevant-nonrelevant, nonrelevant-nonrelevant).

Figure 8 shows there is sometimes a considerable overlap in the cosine similarity distribution for all three relations. The distribution of similarities varies drastically between topics. For the "hz-older-adults" and "beta-thalassemia-genotype-distribution" topics we observe a peak in the relevant-relevant distribution discernable from the relevant-nonrelevant distribution, indicating a higher average similarity between relevant-relevant document pairs. This relation is hardly or not observed in the "meningococcal-outbreaks" and "voedingsadvies-zwangeren" topics,

where a near complete overlap exists between all distributions. This means that for the "beta-thalassemia-genotype-distribution" and "hz-older-adults" topics, there is a stronger correlation between document cosine similarity and relevance when compared to the "meningococcal-outbreaks" and "voedingsadvies-zwangeren" topics. The cosine similarity between document pairs may therefore not always be a reliable indicator for relevance. Even in studies where the distributions are more distinguished we cannot determine a hard cutoff: we cannot decide with certainty that a document with a cosine similarity of 0.9 to a relevant document is itself indeed relevant, or that a document with a similarity less than 0.25 to a relevant document is not relevant.

Another limitation is that we do not know the relevance labels of each document during labeling and can therefore not make such a graph showing clear cut-offs per topic in advance. We cannot set a similarity threshold where, for example, all documents with a similarity below this threshold are not relevant, since this number differs quite a bit between the various topics. However, this graph still gives an important insight and shows us that generally speaking, the similarity between relevant document pairs is somewhat higher than between relevant-nonrelevant document pairs, which is why taking the cosine similarity of papers into account could improve results in some cases.

5 Methods

In this section, we describe our methods and implementation details of the models we will use. These models can both create rankings stand-alone, but can also be integrated in Selectical. First we describe how we preprocessed the data. Then we describe how we test all rankers and calculate their performance. After these necessary introductions, we will describe the implementation of the different models.

5.1 Data processing

Due to the wide variety of the used models in this thesis and the differences in the type of data they require as input, we have various methods of data processing.

The title and abstract of all papers were appended for each model. In one implementation of sentenceBERT, the title and abstract were split in separate sentences in order to be able to compute the similarity between each query and document sentence. The sentences of the document title and abstract were split by using nltk [LB02].

The field specifiers (title, abstract, MeSH, keywords, etc.) were removed from the query for each method, since our data does not contain most of these fields. For each BERT-based model and lexical ranker we use, the Boolean operators (AND, OR, NOT) are removed from the queries. This was done in order to somewhat simulate natural text, and to filter out unnecessary terms that either do not add anything to the meaning of the text. If the topic has multiple queries, or large parts were separated by AND-statements, they were split and each subclause was considered as a separate query sentence. Only for RankFusion CLF we kept the Boolean operators in the query, since this model uses the operators to merge rankings.

For lexical rankers and RankFusion CLF, we experiment with both stemming, lemmatization and stopword removal. For stopword removal, we used the built-in stopword list of Terrier.[6] For stemming, we used the `PorterStemmer` of the nltk library. For lemmatization, we used the `lemma_` function from spaCy [HMVLB20] using the `en_core_web_sm` model, except when lemmatizing the atomic clauses in RankFusion CLF. Here we used nltk's `WordNetLemmatizer`, which uses part of speech tags of nltk's `pos_tag` function. We have chosen for this implementation because the `WordNetLemmatizer` in combination of nltk's `pos_tag` is better than spaCy's `lemma_` when lemmatizing words without context of a sentence, which is what RankFusion CLF uses as atomic clauses.

5.2 Comparing model performance

For each ranker, we let them create a ranking for each topic in our dataset. Even though the dataset has binary relevance labels, each ranker produces a non-binary relevance score for each document in the dataset (except for the supervised Selectical model, which predicts on a smaller test set). These scores allow us to create a ranking for each topic for each ranker.

Considering we want to achieve the highest recall possible with the highest precision, we have chosen to compare the performance of models using the Mean Average Precision or MAP for short. This is the mean of all Average Precisions over all topics. The Average Precision is calculated with both the recall and precision. By continuously lowering the threshold score for documents to consider them relevant, we include more and more documents in the set of documents we consider relevant, which increases the recall. This ,however also results in the precision becoming lower: lowering the threshold increases the number of nonrelevant documents that were incorrectly assigned a high score. The Average Precision is calculated with Equation 6, where R_n and P_n are the recall and precision at threshold n respectively.

$$AP = \sum_n (R_n - R_{n-1}) \cdot P_n \tag{6}$$

Aside from the Mean Average Precision, we also report the minimum Average Precision and maximum Average Precision, as well as the standard deviation. This gives a more complete overview of the performance to expect in worst- and best-case scenarios.

5.3 Supervised and transfer learning using Selectical

Our first attempt involves supervised learning using Selectical. As described before, the final output of Selectical is determined by a simple neural network on which the various features are used as input. Note that due to the lack of relevance labels in real-world scenarios it is often not possible to train (and test) on significant number of papers in a dataset. Only when, for example, one would perform a follow-up study on a topic that Selectical was previously used in, it would be possible to have such a pre-trained Selectical model. These results are to be seen as a snapshot of the model performance when having labeled a certain percentage of randomly selected papers. The training and testing on a single or all topic(s) was mostly done to determine how well performance on topics generalizes, and for comparing performance to the transfer learning performance.

[6] Terrier's stopword list is available at: https://github.com/terrier-org/terrier-desktop/blob/master/share/stopword-list.txt. Retrieved on 13/10/2023.

The small neural network determining Selectical's output can also be used for transfer learning after having trained on one or more studies. Keep in mind that the active learner of Selectical during labeling time resets and is retrained from scratch every time a new paper is labeled, and therefore has to be disabled: otherwise, the network would no longer contain knowledge from previous topics. We decide to test various setups of supervised and transfer learning to compare performance. We used the following setups:

- Training a model on a percentage of papers of one topic, and testing on the other papers of that topic not seen during training. This creates 39 individual models that are all tested separately on their own dataset.
- Training a model on a percentage of papers of all topics, and testing on the other papers of all topics not seen during training. This creates one single model.
- Training a model on all papers of a number of topics, and testing on all papers of the topics not seen during training. Here, we alternate between which topics the model trains and tests on. We have chosen to split this in four parts, so each model would train on 75% and test on 25% of all topics. This creates four models that are all tested on the topics they were not trained on. We have also chosen to train 39 models, each trained on all except one topic, and tested on that single topic.

The first two methods set an absolute upper bound. The first method tells us the performance of the model when having classified a number of papers of a single topic, which is the best performance that can be achieved on that topic. The second method is similar in that sense, but could give insights for how well the model generalizes between topics. We use multiple train and test sizes. Keep in mind that the performance of, for example, a 50% training set is not entirely equal to the performance of Selectical during a labeling session. When choosing the train and test set the papers are chosen randomly, while in real use cases the reviewer will have a better suggested order due to the active learner.

We use the last method to test how well the model generalizes between topics. This method is quite standard in the information retrieval field. It is a form of transfer learning, since we test on different topics than the model was trained on. The model is a zero-shot model for the topics it is not trained on. This is a realistic scenario when performing information retrieval, since a model could never be trained on all topics a researcher would want to retrieve documents for. Note once again that active learning is not be possible for all of the setups above, with the current setup of retraining the MLP from scratch every time.

Considering the small size of the neural networks, we have chosen to train the networks in each of the above configurations for 50 epochs. We use train sets of sizes 10%, 30%, 50%, and 80%, with training sizes of respectively 90%, 70%, 50%, and 20%.

5.4 Lexical rankers and RankFusion CLF

We have chosen to implement the lexical rankers and RankFusion CLF using PyTerrier [MT20], a Python package that consists of bindings to the Java-based Terrier library [MMSO12]. Terrier is an indexing library which offers a multitude of lexical ranking models, including BM25, TF-IDF,

DPH, InL2, and more.[7] Note that the default BM25 implementation of Terrier does not have the +1 term described in Section 2.5. Hence we use a variation called `BM25_log10_nonum` which does include this term.[8] Additionally, the IDF ranker is currently implemented differently from other lexical rankers in Terrier, even though both TF and TF-IDF are available. Since TF-IDF = TF × IDF, we worked around this by calculating IDF = $\frac{TF-IDF}{TF}$. InL2 is a lexical ranker based on IDF and uses Laplace normalization.

We want to compare the performance of the lexical rankers on their own to the performance of RankFusion CLF to determine whether the fusion method has a positive impact on the ranking quality. Since some of our queries consist of multiple subqueries, each subquery assigns a score to each document and we sum these to create a final ranking. This way, documents scoring high in multiple subqueries get higher final scores.

For RankFusion CLF, we closely follow the description of the original implementation. However, our data lacks some of the features used by the original RankFusion CLF, so we are unable to implement it exactly the same way as the authors. Our closest approximation uses a combination of BM25, TF-IDF, InL2 and IDF since these are the lexical rankers used by RankFusion CLF that are also available to us.

5.5 BERT-based models

Since BERT-based models are not optimized for Boolean input text, we experimented with various input methods to find which perform best. We have not fine-tuned any transformer model to the general biomedical domain or to a topic of our dataset. Previous experiments have shown that fine-tuning transformer models takes large amounts of data before the model outperforms BM25 [LNY22]. The size of our dataset is currently too limited to reach acceptable performance.

5.5.1 SentenceBERT: bi-encoders

For SentenceBERT (bi-encoders), we use the SentenceTransformers library.[9] The SentenceBERT model we use is a bi-encoder called `all-mpnet-base-v2`[10], which was pre-trained on 1 billion sentence pairs. It has a maximum input length of 384 tokens. Note that both the query and document can have this number of tokens. If the input is longer, it is truncated. We have three different bi-encoder setups:

1. Encoding each query sentence separately and each document sentence separately. We then calculate the similarity between each query sentence and document sentence. If a query has m sentences and a document has n sentences, each document has $m \times n$ similarities. Note that while m is constant in a single topic, n differs per paper. Of all these $m \times n$ similarities, a single number is taken: the average or the maximum, which denotes the score for that document. We refer to this method as "simple".

[7] A list of available rankers is available at: http://terrier.org/docs/current/javadoc/org/terrier/matching/models/package-summary.html.

[8] Implementation available at: https://github.com/terrierteam/terrier-ciff/blob/master/src/main/java/org/terrier/matching/models/BM25_log10_nonum.java

[9] https://www.sbert.net/examples/applications/cross-encoder/README.html. Date accessed: 15-05-2023.

[10] https://huggingface.co/sentence-transformers/all-mpnet-base-v2. Date accessed: 21-12-2023.

2. Encoding each query sentence separately and encoding the document as a whole. We now calculate m cosine similarities per document, as there is no variable n number of sentences. The score of a document is the sum of these m similarities. We refer to this method as "wholetext".

3. Encoding each query sentence separately and encoding each document sentence separately. For each query sentence we calculate its similarity to all document sentences, and take the average or maximum similarity. This gives us m similarities. The score of a document is the sum of these m similarities. We refer to this method as "per sentence".

Generally speaking, we used the maximum or the average similarities between query and document. Both are intuitive: if we score a document by its maximum similarity to the query, we logically find the most relevant part of the document to that query. Scoring a document by its average similarity to a query is a more nuanced approach where we want multiple parts of a document to be relevant to the query, rather than a single part.

We have chosen to not encode the queries as a whole due to the great length some queries have. These do not fit as a whole in any BERT model. Recall that each query sentence is actually a processed Boolean sub-query. Each Boolean sub-query has a different aspect it tries to retrieve, and therefore, we would lose information if latter sub-queries would be truncated due to BERT input size limitations, hence we have chosen to only encode the queries in parts.

Note that the "simple" method condenses everything in a single number and might therefore lose information. The "wholetext" method has the advantage of having the whole document in its encoding, which should help with contextual understanding. However, it might suffer from less redundancy due to having less similarities to work with. The last method, "per sentence", is a hybrid of both. Note that both the "wholetext" and "per sentence" method produce a $m \times d$ matrix of similarities, where d is the number of documents in the dataset (for that study), before being processed into a single score. This $m \times d$ matrix of similarities could also in theory be processed in some other way, for example, to train a neural network.

5.5.2 Cross-encoders

We used the SentenceTransformers library to implement cross-encoders as well. We have chosen to use the two best-performing cross-encoders[11] pre-trained tasks similar to ours. **1.** The `ms-marco-MiniLM-L-12-v2`[12] model, which was trained on real queries of the Bing search engine. **2.** The `stsb-roberta-large`[13] model, which was trained to indicate the semantic similarity between sentence pairs. Both cross-encoders have a maximum input length of 512 tokens in which both the query and document have to fit. If any or both is longer, the query and document are truncated by longest first. The output of the cross-encoders was mapped between [0, 1] using a sigmoid function, where 0 is not similar in meaning and 1 is exactly the same meaning. We encoded the title and abstract of the documents as a whole with each separate query sentence since encoding each document sentence with each query sentence would be too computationally expensive. The final score is calculated by summing the all document scores for each query sentence.

[11] https://www.sbert.net/docs/pretrained_cross-encoders.html Date accessed: 25-01-2024
[12] https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2. Date accessed: 21-12-2023.
[13] https://huggingface.co/cross-encoder/stsb-roberta-large. Date accessed: 21-12-2023.

5.5.3 ColBERT

Our preprocessing steps for ColBERT only slightly differ from those of SentenceBERT. We still split our Boolean queries into sentences. However, we do not split the document into separate sentences. We have chosen to use the ColBERT implementation offered in `pyterrier-colbert` [WMTO21]. This is a plugin package based on the PyTerrier library, also described in Section 5.4. For each query sentence each document gets a score. The final score is calculated by summing the all document scores for each query sentence.

6 Experiments and Results

This Section describes our experiments in further detail and the results we obtained with our experiments. The best results in tables are marked in bold per metric. The hardware used can be found in Appendix A.

6.1 Supervised and transfer learning using Selectical

We first describe the performance of Selectical when trained as a supervised model. The results are displayed in Table 1. The columns denote the Mean Average Precision, its standard deviation, the Minimum Average Precision and the Maximum Average Precision. These last two metrics are the Average Precision of the topic on which these models performed the worst and best, respectively. These may differ per model and are shown to indicate the range of performance one can expect.

Selectical-IR denotes the model that was trained and tested on alternating parts of the dataset. LOO stands for **L**eave **O**ne **O**ut, meaning that Selectical was trained on all topics except one, and tested on that dataset. Split4 means the dataset was split in four parts, training on 75% of topics and testing on 25%. Selectical-all refers to the model trained and tested on all topics. Selectical-individual is the model that was trained and tested on a single topic. tr-0.X denotes the size of the train set and intrinsically the size of the test set, which is always $1 - 0.X$. For example, tr0.3 means a train size of 30% and a test size of 70%. Keep in mind that the size of the test set changes with the size of the train set, and that there will be less documents to classify if the train set gets larger. All models were trained for 50 epochs, as denoted by ep50.

As expected, Selectical-individual performs best since its test set concerns the same topic as its train set. The model is therefore "specialized" in that single topic. The performance of this model can be seen as an upper bound. When training and testing on all topics using the Selectical-all model, it seems that the model is able to classify papers of all topics as well, although the performance is substantially lower than models specifically trained on a single topic. The Selectical-individual model trained with a train size of 10% on a single topic has a higher MAP than the Selectical-all model trained on 50% of all papers across all topics.

We see that training on an alternating set of 75% of all topics and testing on the other 25% has the worst performance of all supervised models. This is not unexpected because the model has not seen any papers of the test topics. The Leave One Out model also performs worse than all other models, even though it was trained on more papers than all other models. We observe that Selectical does not generalize well on all the topics at all.

Model	mean	std	min	max
selectical-IR-LOO_ep50	0.173	0.161	0.028	0.682
selectical-IR-split4_ep50	0.167	0.149	0.027	0.580
selectical-all_ep50-tr0.1	0.270	0.202	0.048	0.701
selectical-all_ep50-tr0.3	0.327	0.208	0.035	0.766
selectical-all_ep50-tr0.5	0.372	0.225	0.062	0.822
selectical-all_ep50-tr0.8	0.410	0.248	0.040	0.870
selectical-individual_ep50-tr0.1	0.373	0.216	0.024	0.733
selectical-individual_ep50-tr0.3	0.489	0.194	0.176	0.797
selectical-individual_ep50-tr0.5	0.541	0.180	**0.210**	0.853
selectical-individual_ep50-tr0.8	**0.613**	0.213	0.183	**1.000**

Table 1: Average Precision of Selectical in supervised set-ups

6.2 Lexical rankers and RankFusion CLF

We now discuss models that can be used in zero-shot scenarios. First, we describe the performance of lexical rankers on their own, after which we combine them using the RankFusion CLF method.

6.2.1 Lexical rankers

We first discuss the results of the lexical rankers used in the original RankFusion CLF paper: IDF, TF-IDF, BM25 and InL2. All four were tested with and without both stopword removal, stemming, or lemmatization. The results for the IDF-based ranking can be seen in Table 2. Table 3 shows the results of the TF-IDF ranker, Table 4 the results of the BM25 ranker (the log10_nonum variant of Terrier), and Table 5 the results of the InL2 ranker. Stopwords-no denotes the stopwords have been removed, while stopwords-yes means that they were left in. The last part of the model name tells whether lemmatization, stemming, or neither was applied.

Model	mean	std	min	max
IDF_stopwords-yes_None	0.143	0.115	0.022	0.416
IDF_stopwords-yes_lemmatization	0.154	0.130	0.024	0.482
IDF_stopwords-yes_stemming	0.143	0.117	0.022	0.438
IDF_stopwords-no_None	0.144	0.118	0.022	0.440
IDF_stopwords-no_lemmatization	**0.156**	0.130	**0.025**	**0.489**
IDF_stopwords-no_stemming	0.144	0.119	0.022	0.441

Table 2: Average Precision of IDF-based ranking.

For the IDF-based ranker, we observe that removing the stopwords from queries and documents results in a higher performance. Stemming the words does not seem to have a large impact the performance. However, using lemmatization increases the performance.

TF-IDF performs much better than just using IDF, which is expected. We notice the same observations concerning the preprocessing steps when looking at the results of TF-IDF. Again, removing stopwords increases the performance slightly. Using lemmatization again results in the best performance. Stemming has little to no impact.

Model	mean	std	min	max
TF-IDF_stopwords-yes_None	0.214	0.161	0.022	0.610
TF-IDF_stopwords-yes_lemmatization	**0.233**	0.177	**0.025**	0.623
TF-IDF_stopwords-yes_stemming	0.213	0.163	0.022	0.608
TF-IDF_stopwords-no_None	0.217	0.163	0.022	0.603
TF-IDF_stopwords-no_lemmatization	**0.233**	0.179	**0.025**	**0.632**
TF-IDF_stopwords-no_stemming	0.217	0.166	0.022	0.600

Table 3: Average Precision of TF-IDF ranker.

Model	mean	std	min	max
BM25_stopwords-yes_None	0.207	0.162	0.022	0.579
BM25_stopwords-yes_lemmatization	**0.224**	0.177	**0.023**	0.602
BM25_stopwords-yes_stemming	0.206	0.164	0.022	0.577
BM25_stopwords-no_None	0.210	0.163	0.022	0.573
BM25_stopwords-no_lemmatization	0.223	0.178	**0.023**	**0.621**
BM25_stopwords-no_stemming	0.210	0.165	0.022	0.571

Table 4: Average Precision of BM25 ranker (log10_nonum variant of Terrier).

Model	mean	std	min	max
InL2_stopwords-yes_None	0.205	0.160	0.022	0.575
InL2_stopwords-yes_lemmatization	**0.223**	0.176	**0.023**	0.599
InL2_stopwords-yes_stemming	0.205	0.162	0.022	0.573
InL2_stopwords-no_None	0.209	0.162	0.022	0.571
InL2_stopwords-no_lemmatization	0.222	0.177	**0.023**	**0.617**
InL2_stopwords-no_stemming	0.208	0.165	0.022	0.569

Table 5: Average Precision of InL2 ranker.

BM25 surprisingly performs slightly worse than TF-IDF. Here, not removing stopwords gives slightly higher performance. However, applying lemmatization still has a beneficial effect. Stemming again does not seem to affect performance substantially.

InL2 performs very similarly to BM25, and also performs the best using the same preprocessing settings: not removing the stopwords, and applying lemmatizations. Once again, using stemming does not affect performance substantially.

6.2.2 RankFusion CLF

We now inspect the performance impact of RankFusion CLF over the solo lexical rankers. We combined all lexical rankers used in the original RankFusion CLF. The performance of these lexical rankers in the RankFusion CLF setup is displayed in Table 6. In this table, we have abbreviated "stopwords" to "stop", "lemmatization" to "lemma" and "stemming" to "stem".

Model	mean	std	min	max
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-yes_None	**0.225**	0.167	0.024	**0.668**
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-yes_lemma	**0.225**	0.168	0.024	0.664
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-yes_stem	0.222	0.168	0.024	0.663
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-no_None	0.219	0.167	**0.029**	0.661
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-no_lemma	0.221	0.168	**0.029**	0.661
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-no_stem	0.219	0.167	**0.029**	0.653

Table 6: Average Precision of RankFusion CLF ranker using all previously tested lexical rankers.

RankFusion CLF does not perform very differently from the individual lexical rankers. Interestingly, the impact of using lemmatization and stemming on the ranker's performance is diminished in RankFusion CLF. The removal of stopwords has a slightly negative impact on performance.

6.3 SentenceBERT: bi-encoders

After the performance of lexical-based models, we now investigate transformer models. We first take a look at our various set-ups for bi-encoders. The results can be seen in Table 7.

The simple method with the maximum similarity performs the worst of all methods. the simple average and per-sentence average methods perform somewhat better and, interestingly, very similarly. The per-sentence maximum method performs only slightly better on average, but has the largest standard deviation. We see that the wholetext method performs the best, having both the highest mean, maximum and minimum average precision. It seems that the bi-encoder is able to take advantage of context, since it is the only model that has the whole text encoded in a single embedding, rather than separate embeddings for each sentence.

All methods have a standard deviation that is quite high. The variance in performance is also reflected in the very low minimum average precision and relatively high maximum average precision.

Model	mean	std	min	max
bi-encoder_allmpnetbasev2_simple-avg	0.236	0.182	0.025	0.684
bi-encoder_allmpnetbasev2_simple-max	0.179	0.156	0.011	0.661
bi-encoder_allmpnetbasev2_wholetext	**0.272**	0.196	**0.041**	**0.750**
bi-encoder_allmpnetbasev2_per-sentence_avg	0.236	0.180	0.025	0.684
bi-encoder_allmpnetbasev2_per-sentence_max	0.238	0.199	0.031	0.738

Table 7: Average Precision of various bi-encoder setups.

6.4 Cross-encoders

We now take a look at the performance of various cross-encoders. Their results are displayed in Table 8. The cross-encoders perform worse than the bi-encoders. The `ms-marco-MiniLM-L12-v2` model is able to approximate the performance of some of bi-encoder setups, however `stsb-roberta-large` performs much worse than all bi-encoders. We suspect the reason

for this worse performance is due to the smaller input length the cross-encoders can handle. Both cross-encoders have an input size of 512 in which both the query and document have to fit. Considering that some of our queries are much, much longer than this input size, the document gets truncated as well, therefore losing information.

Model	mean	std	min	max
cross-encoder-ms-marco-MiniLM-L-12-v2_512	**0.222**	0.176	**0.030**	**0.630**
cross-encoder-stsb-roberta-large_512	0.118	0.106	0.019	0.419

Table 8: Average Precision of two cross-encoder architectures.

6.5 ColBERT

The results of ColBERT can be seen in Table 9. `ColBERT` also performs worse than most bi-encoder setups, and similarly to the `ms-marco-MiniLM-L12-v2` cross-encoder. We suspect that ColBERT might suffer from a similar problem as our per-sentence bi-encoder, where not encoding queries and documents as a whole leads to a lesser understanding of context and therefore to a lesser performing ranker.

Model	mean	std	min	max
colbert	0.214	0.173	0.026	0.688

Table 9: Average Precision of ColBERT.

6.6 All tested zero-shot models

In this subsection, we compare the best performing parameters for each model. We use the MAP to determine which parameter set performed best. If those are equal, we look at the minimum and maximum AP as well. The results can be seen in Table 10. Precision-Recall curves for these best-performing models are shown in Figure 14 in Appendix A.

Model	mean	std	min	max
IDF_stopwords-no_lemmatization	0.156	0.130	0.025	0.489
TF-IDF_stopwords-no_lemmatization	0.233	0.179	0.025	0.632
BM25_stopwords-yes_lemmatization	0.224	0.177	0.023	0.602
InL2_stopwords-yes_lemmatization	0.223	0.176	0.023	0.599
rankfusionclf_IDF_TF-IDF_BM25_InL2_stop-yes_None	0.225	0.167	0.024	0.668
bi-encoder_allmpnetbasev2_wholetext	**0.272**	0.196	**0.041**	**0.750**
cross-encoder-ms-marco-MiniLM-L-12-v2_512	0.222	0.176	0.030	0.630
colbert	0.214	0.173	0.026	0.688

Table 10: Average Precision of all tested zero-shot models with the combination of parameters achieving the highest performance.

The bi-encoder outperforms all other models in every metric. Among the lexical rankers we see that lemmatizing query and documents gives the best performance in all cases, except when using RankFusion CLF. Stopword removal seems to have a positive impact on performance for the simpler lexical models (IDF and TF-IDF), while leaving stopwords in the texts results in better performance for the more complex models (BM25 and InL2).

Stemming generally did not seem to have a meaningful impact on performance, while lemmatization improved performance in all standalone lexical rankers. This could be caused by the fact that lemmatization reduces inflections of words to their lemma, i.e., the actual base form of a word. Stemming, on the contrary, reduces words to a stem, which may or may not be an actual existing word. Stemming may therefore result in some words with the same meaning not being matched, or words with different meanings that are wrongly matched. For example, "caring" and "cared" would be stemmed to "car", which causes semantic mismatches. In addition, "am", "be", "are", etc. are not stemmed to "be" and will not result in matches using lexical rankers. Using lemmatization would work in these cases: "be" is the correct lemma for these inflections, and "care" is the lemma of "caring" and "cared". Therefore, stemming may have a limited impact overall: sometimes it could lead to more correct matches, while in other cases it will result in more mismatches. Lemmatization on the other hand improves the performance somewhat due to the lexical rankers being able to exact-match more words thanks to the common and correct lemma.

In general, the lexical rankers were hampered by the fact that most Boolean queries contain some wildcards. For example, achondroplas* in a query represents both achondroplastic and achondroplasia due to its wildcard. These terms with wildcards do not always have a correct exact match, and lemmatizing or stemming may not result in a usable form that has a match in documents, limiting the impact of both stemming and lemmatization.

The best lexical ranker is, surprisingly, TF-IDF, although all lexical rankers except IDF have similar performance. Both ColBERT and the cross-encoders are outperformed by all lexical rankers, except IDF. The cross-encoder comes close to lexical ranker performance, however, has a much longer compute time than the bi-encoder [RG19] and any lexical ranker, whose computational time is very short: lexical rankings are calculated using relatively simple equations. Based on these results, we suggest using either a bi-encoder for the best performance, or a lexical ranker for the fastest speed for this particular dataset.

Cross-encoders might outperform bi-encoders if their input size were larger than the current 512 tokens. However, the pre-trained transformer models of the SentenceTransformers library are limited to a maximum size of 512. To use a larger input size we would have to train our own cross-encoder, which is very computationally expensive and requires a large amount of data.

7 Integration in Selectical

The previous section compared the performance of various stand-alone ranking models. In this section we attempt to integrate our previous methods in Selectical to attempt to improve its performance. We first describe how we evaluate the performance of Selectical. Then we introduce two integration methods: the first uses a ranker from the previous section to initialize Selectical's MLP to mitigate the cold-start problem. The second method is based on the intuition that relevant seed documents help with finding other relevant documents and adds a new active learning component to Selectical.

7.1 Evaluation

We measure the performance of Selectical using the labeling load. This is measured with the average and weighted average percentage of papers that had to be labeled to achieve a desired recall. The lower the percentage, the less papers had to be labeled. The weighted average is weighed by the number of papers in a topic, so a reduction in number of papers to be labeled for a large set has a higher impact on this metric than a reduction on a small set. We test labelling load for two recall levels R: $R = 100\%$ and $R = 99\%$. We define n_R as the number of relevant papers to be found to achieve the desired recall R. Of course for $R = 100\%$ we define $n_{100\%} = \#rel$, where $\#rel$ is the number of relevant documents for a certain topic. For $R = 99\%$ the definition of $n_{99\%}$ is slightly more complex, as $n_{99\%} = 0.99 \times \#rel$ will often not result in an integer. We therefore always round n_R up to the nearest integer. However, since some topics have less than 100 relevant papers, for example a topic where $\#rel = 68$ we would have $n_{99\%} = \lceil 0.99 \times 68 \rceil = \lceil 67.32 \rceil = 68$, resulting in $n_{100\%} = n_{99\%} = \#rel$. To prevent this we define $n_{99\%} = min(\lceil 0.99 \times \#rel \rceil, \#rel - 1)$. In the previous example, $n_{99\%} = min(\lceil 0.99 \times 68 \rceil, 68 - 1) = min(\lceil 67.32 \rceil, 67) = min(68, 67) = 67$. For a topic with $r > 100$, for example, $r = 147$, $n_{99\%} = min(\lceil 0.99 \times 147 \rceil, 147 - 1) = min(\lceil 145.53 \rceil, 146) = min(146, 146) = 146$. For a topic with $r >> 100$, for example, $r = 372$, $n_{99\%} = min(\lceil 0.99 \times 372 \rceil, 372 - 1) = min(\lceil 368.28 \rceil, 371) = min(369, 371) = 369$.

We calculate the labeling load by simulating a labeling session using Selectical for each tested setup. This works by continuously retraining the MLP at the end of the model architecture, as described in Figure 2. When used for actual labeling sessions, Selectical will retrain after each labeled paper, however, to speed up the simulation this is done less frequently. When papers are labeled they are added to the train set. During simulation we assume a perfect user who labels each paper with its known label. The training steps can be seen in Table 11. The step size denotes the number of new papers that need to be labeled before being retrained. The maximum step denotes the maximum number of papers that are labeled and added to the train set before the model switches to the following training setting. The batches denote how many times the model will retrain for that training setting. For example in the first setting, Selectical will retrain each time 5 new papers are labeled a total of 20 times, after which it will have learned on 100 papers. In the next step, Selectical will retrain every 10 new labeled papers for 40 times, after which it will have learned on 400 new papers, etc. Note that Selectical originally also has an early stopping criterion which will, theoretically, notify the labeling agent when it is likely no more relevant papers can be found. Early stopping can only trigger whenever at least 150 papers or 15% of all papers have been labeled, whichever is more. This is done to ensure Selectical has a substantial number of training samples to learn paper relevancy. If this requirement is satisfied, early stopping triggers when a second condition is met: if in the last x labeled papers no relevant papers were found, early stopping triggers. The intuition is that when no relevant paper has been found in the last x labeled papers, there are likely no more or only a few relevant papers left since the model cannot find any more relevant papers. x is variable and changes according to Equation 7, where "nlabeled" is the total number of papers that has been labeled so far.

Step size	5	10	25	50	100	200
Max Step	100	400	500	1000	1000	no limit
Batches	20	40	20	20	10	no limit

Table 11: The training step sizes and number of batches for that step size, as well as the maximum number of papers for each step size the model is trained on. The model is trained less and less frequently. Steps of size 200 in the last stage are repeated indefinitely until all papers have been labeled.

$$\text{Stop if no relevant paper in last: } max(x, 1\% \times \text{nlabeled}) \begin{cases} x = 140 & \text{if nlabeled} < 500 \\ x = 70 & \text{if } 500 \geq \text{nlabeled} < 1500 \\ x = 35 & \text{if nlabeled} \geq 1500 \end{cases} \tag{7}$$

The intuition behind the stopping criterion becoming stricter is that Selectical should learn to discern which papers are relevant and which ones are not. In the beginning, when Selectical has not yet trained on a large set of papers, the stopping criterion is more lenient, while it gets more strict the more papers have been labeled and trained on. However, since some topics are much larger, decreasing the early stopping may result in the stopping triggering too early. The number of papers we have labeled so far is therefore also taken into account, which increases the stopping criterion if we have labeled a lot of papers, i.e., are labeling a very large dataset. In our experiments, we implemented the stopping criterion by not retraining the MLP anymore whenever the stopping criterion was reached and ranking the remaining papers with the last trained model.

7.2 Alleviating the cold-start problem

The simplest way of integrating one of the previous methods with Selectical is by letting a model determine the first N texts to be labeled by the reviewer, rather than the active learner which has not learned anything yet. We kickstart the active learner with computationally cheap models of the previous section. These rankers provide at least some relevant documents in these N proposed texts and theoretically, the active learner should be able to identify relevant documents quicker if it was initialized on multiple relevant examples. After these N papers, the active learner takes over the selection process. The training steps from Table 11 are respected from then on. We have chosen to initialize on either 50, 100 or 200 papers to determine which N gives the best performance. Considering that the `all-mpnet-base-v2` bi-encoder has the highest Mean Average Precision using the "wholetext" setup, we use this model in this setup in all cases. We compare Selectical initialized with the first N papers retrieved by the bi-encoder to a setup where Selectical is not initialized on anything and the active learner immediately tries to present the reviewer the most relevant paper. The results can be seen in Tables 12 and 13. Here, "Selectical-no-init" is Selectical in its original state without our improvements, and Selectical-initN denotes the number N of papers Selectical was initialized on using the bi-encoder. Note that for the "init200" setting, one topic containing less than 200 papers was completely labeled in the order presented by the bi-encoder since the initialization size was larger than the number of documents for that topic. Selectical did not perform any labeling during the simulation for that single topic.

Model	average	weighted average
Selectical-init50	81.2%	**77.6%**
Selectical-init100	**77.8%**	78.3%
Selectical-init200	78.3%	79.4%
Selectical-no-init	81.8%	79.9%

Table 12: Results in terms of labeling load of different initialization strategies for 100% recall. Lower numbers indicate a more effective ranker.

Model	average	weighted average
Selectical-init50	67.8%	**64.9%**
Selectical-init100	66.9%	66.8%
Selectical-init200	**66.3%**	65.3%
Selectical-no-init	70.8%	68.1%

Table 13: Results in terms of labeling load of different initialization strategies for 99% recall. Lower numbers indicate a more effective ranker.

For achieving a 100% recall, initializing on 100 papers gives the least labeling load on average. For the weighted average the results of all setups are closer to each other with a slight edge for initializing on just 50 papers. For 99% recall initializing on 200 papers gives the best performance, but only by a slim margin. For the weighted average, initializing on just 50 papers has the best performance once again.

7.3 Adding an active learning component

Relevant documents have on average a slightly higher similarity with other relevant documents, as seen in Figure 8. As suggested by literature in Section 3, cosine similarity is a straightforward way of measuring textual similarity between query and document. Previously, we have described our implementation where we attempt to use these similarities to determine document relevance. However, as suggested by [LS18], one can also use the similarity between seed documents and the documents that are yet to be labeled. This also has another added benefit: while the cosine similarity between query and document is a strong and logical way to assess the relevance of a document to the query, it may occur that the query was poorly formulated. In this case, using seed documents may solve this problem: instead of looking for documents similar to a mismatched query, we are searching for documents similar to known-relevant documents. We have therefore chosen to implement an extra component to Selectical which has as feature the cosine similarity between the current to be labeled document and all other documents that were already labeled. Where previous attempts using document similarity chose to use TF-IDF and word2vec embeddings [LS18], we choose to implement this using the same `all-mpnet-base-v2` bi-encoder we used for query-document similarity. In addition, rather than calculating the cosine similarity between the seed documents and the yet-to-be-ranked documents once, we have chosen to implement an active learning component for Selectical which is able to use the cosine similarities during ranking.

Our implementation continuously updates a matrix of cosine similarities. For each document, a set of statistics for cosine similarity are calculated between the document and all already-ranked documents, namely:

- The similarity between the current document and the most similar already labeled relevant document.
- The similarity between the current document and the least similar already labeled relevant document.
- The average similarity between the current document and all already labeled relevant documents.
- The similarity between the current document and the most similar already labeled nonrelevant document.
- The similarity between the current document and the least similar already labeled nonrelevant document.
- The average similarity between the current document and all already labeled nonrelevant documents.

These six numbers for each document are fed in an additional head, in the same way as described in Section 2.3. Contrary to the other static features used by Selecical, this feature changes over time and is continuously updated whenever new documents are added to the set of already labeled documents. By continuously updating these numbers, the model can learn to look for documents similar to already-labeled relevant documents. We combine the cosine similarity with the bi-encoder initialization method from the previous subsection. When not using the bi-encoder to initialize, the cosine similarities are set to 0 during the first training step, since there are no past papers that were labeled on. The results can be seen in Tables 14 and 15. We also include the results of our experiments without the cosine similarity component, so we can compare all setups. Setups with the cosine similarity component enabled are denoted with "cossim". The full tables containing the labeling loads for each topic using each setup can be found in Tables 17 and 18 in Appendix A.

Model	average	weighted average
Selectical-cossim-init50	80.7%	78.7%
Selectical-cossim-init100	79.7%	79.2%
Selectical-cossim-init200	**77.1%**	77.5%
Selectical-cossim-no-init	81.7%	**77.0%**
Selectical-init50	81.2%	77.6%
Selectical-init100	77.8%	78.3%
Selectical-init200	78.3%	79.4%
Selectical-no-init	81.8%	79.9%

Table 14: Results in terms of labeling load of different initialization strategies combined with active cosine similarity leaner for 100% recall. Lower numbers indicate a more effective ranker.

When achieving a recall of 100%, initializing on 200 papers and using the cosine similarity component has the least average labeling load, while not initializing on any papers has the best

Model	average	weighted average
Selectical-cossim-init50	67.9%	67.0%
Selectical-cossim-init100	67.1%	65.9%
Selectical-cossim-init200	**64.3%**	**63.6%**
Selectical-cossim-no-init	69.6%	67.1%
Selectical-init50	67.8%	64.9%
Selectical-init100	66.9%	66.8%
Selectical-init200	66.3%	65.3%
Selectical-no-init	70.8%	68.1%

Table 15: Results in terms of labeling load of different initialization strategies combined with active cosine similarity leaner for 99% recall. Lower numbers indicate a more effective ranker.

Model	rel. fnd	nrel. fnd	rel. miss	nrel. miss	labeled	not labeled
Selectical-cossim-init50	92.8%	35.1%	7.1%	65.0%	58.9%	41.1%
Selectical-cossim-init100	93.0%	35.2%	6.9%	64.8%	58.6%	41.4%
Selectical-cossim-init200	92.3%	32.7%	7.6%	67.3%	55.5%	44.5%
Selectical-cossim-no-init	93.4%	35.9%	6.6%	64.1%	59.6%	40.4%
Selectical-init50	92.1%	35.1%	7.8%	64.9%	58.4%	41.6%
Selectical-init100	92.5%	34.8%	7.4%	65.3%	57.5%	42.5%
Selectical-init200	91.4%	33.5%	8.5%	66.5%	56.3%	43.7%
Selectical-no-init	93.1%	36.5%	6.9%	63.5%	61.0%	39.0%

Table 16: Average statistics at early stopping for each model setup. "rel." stands for "relevant" and "nrel." stands for "nonrelevant". "fnd" stands for "found" and "miss" stands for "missed".

weighted average labeling load when using the cosine similarity component. For a recall of 99%, the best average performance and weighted average performance was achieved by initializing on 200 papers and using the cosine similarity component. However, all model setups perform similarly for both recall levels.

We also investigate the performance of Selectical when using the built-in stopping criterion. Table 16 shows the average results of Selectical over all topics if the user were to stop labeling when the previously described criterion is met. Here, rel. means relevant, nrel. means nonrelevant, fnd. means found, and miss means missed. We note that all models perform very similarly in terms of all metrics. Only slight variations occur, and our methods do not seem to impact the triggering of early stopping, or the recall at that point. Generally speaking, the early stopping triggers on average earlier than when a 99% recall is achieved.

Since all these results are quite close to each other we decided to investigate the labeling process. The labeling load discussed before is a single metric at the end of the labeling session, while the labeling sessions could still progress very differently for each model setup. To investigate this, we calculate the total number of relevant documents that were found during a labeling session at each increment i of one new labeled paper. We show the progression of the recall against the number of labeled documents at each increment i (here shown in percentage of labeled papers) for four topics in Figure 9. The progression of recall for all topics can be found in Figure 15 in Appendix A.

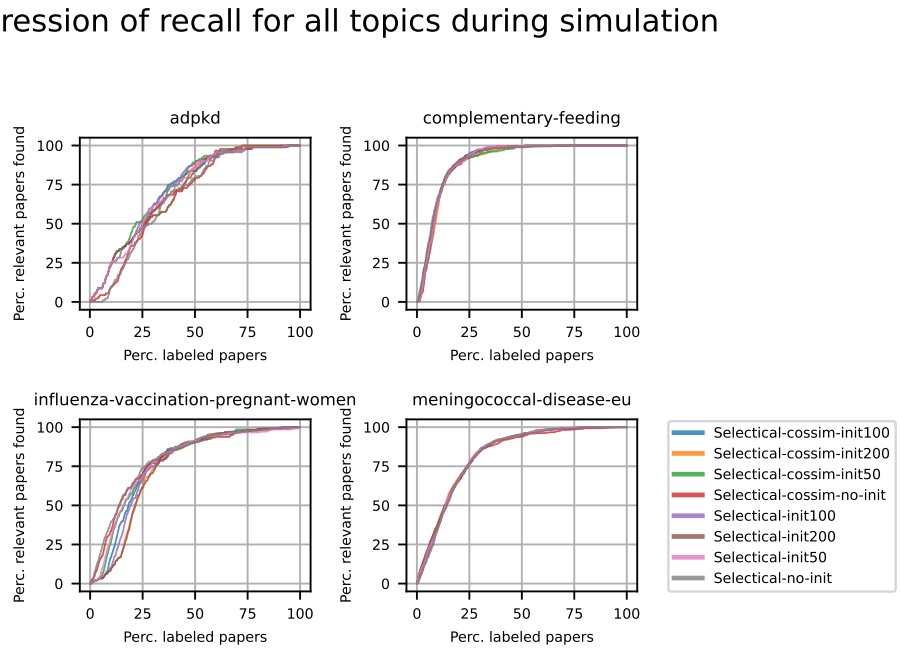


Figure 9: Progression of recall against number of documents that were labeled during simulation of all Selectical setups.

Figure 9 clearly shows all methods have a similar progression of recall. For "complementary-feeding" and "meningococcal-disease-eu", none of our implemented methods seem to have a noteworthy impact on the progression of recall. This is different for the "adpkd" topic, where both setups without bi-encoder initialization trail behind for a large portion of the simulation, although all models converge near the end of the simulation. For "influenza-vaccination-pregnant-women", initializing Selectical using the bi-encoder even seems to have a negative impact, as the "no-init" setups find more relevant documents in the early phases of the simulation. The bi-encoder does not perform well, and Selectical is apparently able to learn fast on its own.

To further compare setup performance, we compare the difference between the number of found documents at each increment i to the baseline ("Selectical-no-init"). The number of papers labeled so far is L. Each topic has P_t papers. We only show the labeling progression of the first 3000 papers (here as absolute numbers), as most topics contain less papers. Note that the number of found relevant papers over all topics are summed. The number of labeled papers is displayed per topic, so an increase of 1 of i is an increase of 1 for each topic, meaning that for these plots a step of 1 is actually an increase of N papers labeled overall, with N being the number of topics that have more papers than L. In later stages, N will be smaller due to some topics having fewer papers: when $L \geq P_t$ for a topic, N will decrease by 1. Note that for these topics having less papers than others, the difference in found relevant papers in the end of their simulation will also become zero for $L \geq P_t$, as both the baseline and the setup we compare it to will have found all relevant papers for that topic when all papers were labeled. The difference between found relevant documents at any point during labeling for each model setup compared to the baseline is displayed in Figure 10.

We observe a measurable difference in number of papers found at the start of labeling sessions. For all methods that initialize using the bi-encoder, we see an advantage during the initial stage.

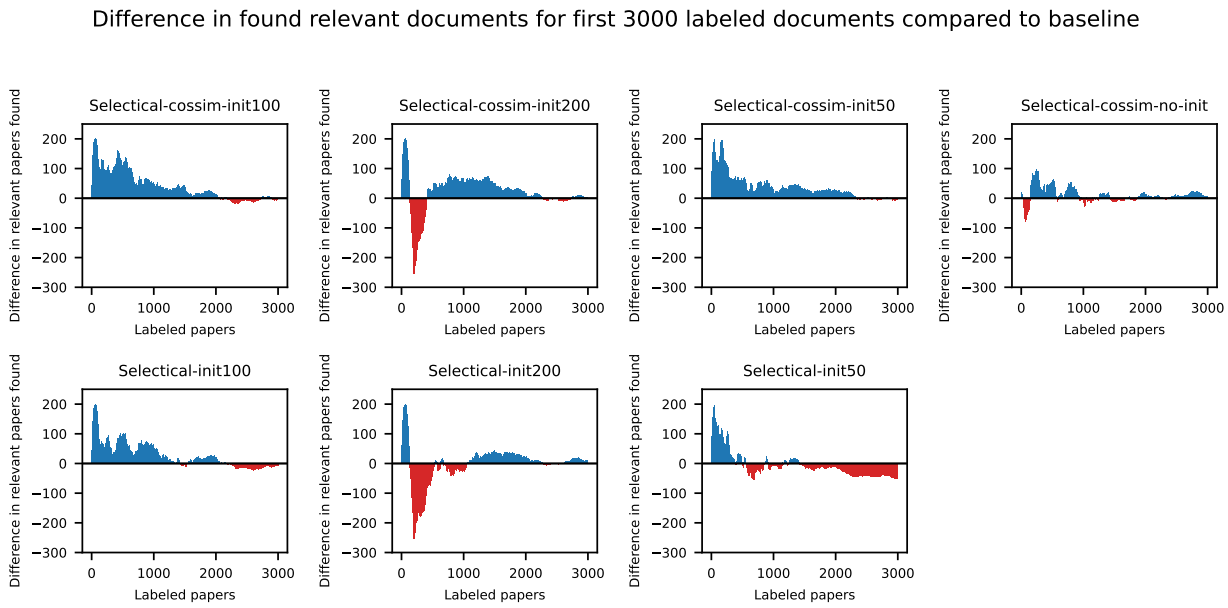


Figure 10: Difference between number of relevant documents found at any step during labeling compared to the baseline "Selectical-no-init".

This advantage is quickly lost for the setup initializing on 200 papers, regardless of the use of the cosine similarity component. This means that the ranking produced by the bi-encoder is already bested by the baseline using the active learner relatively quickly. After sometime, the model initialized on 200 papers is able to recover and gains a small advantage again over the baseline. When initializing using 100 papers, we see a modest advantage that slowly decreases until performance of the baseline is matched after labeling around 2000 papers. The cosine similarity component often does not seem to have a meaningful impact. For initializing on 50 papers, not using the cosine similarity seems to yield a small disadvantage in the long run, while enabling it shows a slight advantage which is less pronounced than when initializing on 100 papers. When not initializing on any papers but using the cosine similarity component, the difference in relevant papers starts slightly negatively, after which a short, slight advantage can be measured, before both setups perform almost identically.

We also show the progression of the Mean Average Precision of Selectical during training. Keep in mind that, contrary to previous shown results, **all documents are reranked** after each training step, and are not taken out of the ranking pool as in usual setups. This means that the ranks of all documents are subject to change, even if these documents have already been labeled in the simulation. While not an entirely accurate representation of the performance of Selectical during labeling, it does give insights in how the model learns over time, and if the overall ranking improves when the model has trained on more papers. The progression of the Mean Average Precision is shown in Figure 11. A plot showing the Average Precision progression graphs for "Selectical-cossim-init100" (according to Figure 10 the best-performing combination) and the baseline "Selectical-no-init" for each topic can be found in Figure 16 in Appendix A. We show both plots in a log-scale x-axis, since we focus on the initial stages. Since some labeling sessions are shorter than others and calculating the MAP over a small set of topics would result in noisy

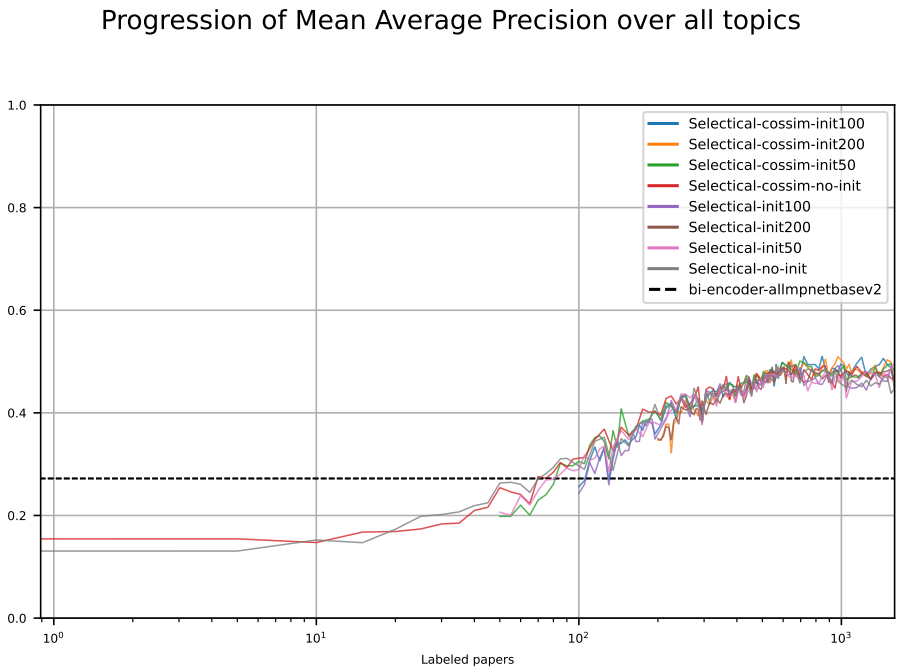


Figure 11: Mean Average Precision progression for all Selectical setups. Note that all papers, including those that were already labeled, are re-ranked after each training step, which differs from the actual setup where labeled papers are taken out of consideration.

results, we cut off the MAP graph when only 5 or less topics are not finished yet. Remember that Selectical has an early-stopping criterion, and that the (Mean) Average Precision will not change anymore after the criterion has been met, even if not all papers have been labeled at that point.

Figure 11 shows that the bi-encoder has a higher performance than Selectical in its initial stages, up until around 100 labeled papers. After this, both the no-init and init50 methods start to achieve a higher MAP. The init100 method matches the performance of the bi-encoder immediately when it takes over from the bi-encoder. The init200 method starts with a slightly lower MAP after taking over than the other Selectical setups, but gets up to the same level of performance rather quickly. Note that when initializing on 200 papers, the quality of the ranking of the bi-encoder is substantially lower between 100 and 200 labeled papers, than the ranking of Selectical models. After this point, all model setups have converged to similar performance. The cosine similarity component has no noteworthy impact on performance.

8 Discussion

Our attempts at improving the performance of Selectical have not been very successful. While some attempts had somewhat promising results for some topics, they failed on others. In this section, we interpret the results of our various methods in an attempt to find possible points of improvement.

8.1 Analysis of negative results

We can clearly see that the last relevant papers are the hardest to find in our experiments determining the labeling load. For all setups, the labeling load is close to and usually more than 10% higher for achieving a 100% recall than when achieving a 99% recall for all setups, even though only a very small number of extra relevant papers need to be found. This is confirmed by various Figures: Figure 15 shows that the curve of found relevant documents flattens for nearly all topics, meaning the last papers require the most effort to find. Figure 11 shows the model's overall rankings do not improve anymore after labeling around 1000 documents, indicating the model is unable to find the last papers automatically. Figure 10 shows that even if you can get a head-start in finding relevant documents using initialization techniques, this advantage will be lost when trying to find all relevant papers. This means that, while we find more relevant papers in the early stages, we only bring those papers forward, rather than shortening the entire selection process. The fact that our implementations for initialization and for the cosine similarity component result in such relatively similar performance may be caused by this same problem: all model runs converge near the end because the last papers are the hardest to find. Since Selectical is unable to rank these last papers effectively, the later stages in the simulation evolve in a similar way as when randomly selecting papers. Differences in results may therefore just be the result of luckier random selections. The fact that some relevant papers are very hard to find may partially be caused by noise in the data in the form of nonrelevant papers that were erroneously labeled as relevant, which would be hard to find for the model.

We attempted to extract information regarding paper relevance by comparing papers to previously labeled papers in the current session. While this is a good idea in theory, we have seen that there is often a considerable overlap in cosine similarity between relevant document pairs and relevant-nonrelevant document pairs. Using the cosine similarity between documents might help in retrieving some initial papers. However, because of this overlap it has a limited or no impact in later stages when retrieving the last relevant papers, which are the hardest to find.

Even if our initial advantage gradually diminishes and we do not reduce the overall labeling load, our methods could improve the user experience. In the past, users of Selectical have stated that the model has a hard time starting to learn, but once they had encountered a few relevant papers and Selectical had trained on them, Selectical would perform much better. Using a bi-encoder to find the first few relevant papers may improve the user experience in this sense, as some of the initially proposed papers should be relevant. We have seen that the bi-encoder has a higher MAP during the initial stages than Selectical, although this advantage does not last very long: using the bi-encoder to retrtieve the first 200 papers already negatively impact performance, and that Selectical usually performs better than or similar to the bi-encoder on its own after having labeled 100 papers.

To answer our first and third research question: None of our attempted implementations, either initializing Selectical using a zero-shot model, or using an extra feature of the cosine similarity between documents, has a large impact on the performance of Selectical in the long run. Our methods were unsuccessful at reducing the labeling load when achieving a recall of 99% or 100%. However, we are able to solve the cold-start problem somewhat, as our bi-encoder initialization method helps with finding the first relevant papers.

8.2 Limitations of the data

We mostly experimented using the Boolean queries in an attempt to determine relevance. This did not result in good performance on all topics. This could be caused by various reasons. For one, we suspect there might be discrepancies between queries and how much useful information they contain. Not all queries may be of good quality. This is supported by Table 10 and Figure 14 in Appendix A. All zero-shot models have a very high standard deviation: the performance of all models varies wildly between topics. We also see some models show promising Precision-Recall curves, while others show an L-shape for all models, indicating bad performance. When inspecting the queries we found some were very short and non-descriptive, while others were extremely long. One short query only described two radioactive isotopes, while some of the very long queries list every large city, region and country in Europe in an OR-statement. This short query is probably too limited and may not even be encoded in any useful embedding by the transformer models, while the latter adds too much noise, decreasing information density.

Another limitation of the queries is that we only had access to the Boolean queries. Although transformer models perform better using natural language, some topics showed promising results using the `all-mpnet-base-v2` bi-encoder. We conclude that encoding Boolean queries using transformer models could be feasible as long as the queries: **1.** Contain more than only a few very specific terms since transformer models need context to create better embeddings; **2.** Are not too cluttered by containing a large number of terms that carry little to no information, or are somewhat cleaned manually to increase information density. To answer our second research question: The best-performing stand-alone model is the `all-mpnet-base-v2` bi-encoder when encoding the document as a whole. This shows that for our dataset, transformer models are able to find relevance information, even with the query not being formulated in natural language.

One should consider that all papers in our dataset were already previously retrieved using said Boolean queries, meaning some or most information might already have been gained out of this feature. Using this same feature once again might not yield much more information. Additionally, the set of retrieved documents is biased towards the Boolean query since the documents were already retrieved by this same query. We are not retrieving any more documents. Using some other or preferably a natural language query or research question might give better results.

8.3 Limitations of this thesis

This thesis has a few limitations. The use of Boolean queries in transformer models is not optimal, and the transformer models may perform better when using natural language. Additionally, we have focused on improving the cold-start problem, while the hardest part of finding all relevant papers is finding the last ones. Another limitation of this thesis is that we have only run the labeling simulation once for each setup. Simulating a labeling session is relatively time-consuming and we were therefore limited in our ability to run the same simulation multiple times to average performance for a certain setup. There could be variations in performance due to random selection in the initial phases, where Selectical without initialization may get more or less lucky in finding its first relevant papers, which would speed up the overall learning process. The difference between Selectical with and without initialization may therefore differ more or less on average. In general, having multiple runs of the same simulation setup would increase the robustness of this study.

8.4 Advances in Large Language Models

During our research and the writing of this thesis, large advancements have been made in natural language processing with the rise of Large Language Models. This includes new generative models such as the GPT-models, like ChatGPT [Ope22] and GPT-4 [Ope23]. While these models are specifically aimed at generating responses based on user input, they have proven to have a very strong understanding of natural language input. Some starting research has been done in the area of implementing LLMs including GPT-based ranking models in information retrieval, either stand-alone or as a labeling assistant [FDC+23]. Attempts at making ChatGPT write Boolean queries have already been made [WSKZ23]. Making a generative model rank papers correctly is not an easy task however. We have tried to ask ChatGPT for relevance feedback on some papers after giving the original Boolean query, but the model claimed each and every paper was relevant. In our understanding, this is not necessarily because GPT is unable to understand Boolean queries: we have also asked ChatGPT to convert Boolean queries to natural text, which worked decently enough on some queries. Rather, we suspect it is because getting ChatGPT to perform a certain task can be challenging, since it is specialized in having conversations. Simply formulating the task as "Tell us whether these papers are relevant or not" is not clear enough as an instruction. GPT probably needs more information on how to determine whether the paper is relevant.

An attempt was made to get GPT models to rank papers [SYM+23]. The authors used various strategies to rank documents. First, they attempted query generation, where the probability that the model generated the original query based on a given paper would determine the relevance of the paper to the original query. Second, they attempted to ask GPT whether a paper was relevant or not, and the relevance score was determined by the probability of the model answering yes or no. The last and best-performing method was permutation generation, where GPT was asked to rank the papers based on relevance in batches. Since GPT has a token limit (although it being much larger than BERT's), not all documents can fit in the input at once, which is why papers were input in batches with a sliding window strategy, i.e., the best or worst papers are also present in the next batch, so they get sorted in a technique somewhat similar to merge sort. The authors claim that GPT was able to outperform MONOT5 and MONOBERT in most datasets. However, the method the authors used is both financially and computationally expensive. The authors acknowledged this and trained a BERT-based model on the output of the GPT model to create a distilled model that would be much cheaper to run, which also performed decently.

As the development Large Language models continues at a fast pace, more research can and should be done in the application of such models in real-world tasks, other than generating output based on user input. Of course, labeling papers with the assistance of such models if an obvious candidate for more research and improvement.

8.5 Possible improvements to our methods

Trying to cluster papers together in some way may result in relevant and irrelevant papers clustering together. We have seen that relevant papers are on average more somewhat more similar to each other than to irrelevant papers. While there is too much overlap in similarity and no clear cut-off can be made, it might be possible to cluster papers together and discard a whole cluster of papers if a certain number of papers in a cluster is considered nonrelevant. However, it is possible Selectical already learns to model relevancy of similar papers using the features currently available.

If going through the route of initializing Selectical using a zero-shot model, one could also add a smarter method of when to switch from the initialization method to the active learner. For example, papers could be put in batches similar to how the simulation of Selectical works. A bi-encoder or some other initialization model used to predict the first documents could then be put against the active learner. If the active learner achieves a higher Mean Average Precision than the bi-encoder when ranking the already labeled papers, Selectical could automatically switch over to the active learner. Another possible improvement would be to use some zero-shot model to determine a set of papers we can definitely exclude before starting the labeling session. However, discarding papers at the start might not actually reduce the labeling load if the automatically discarded papers would already be ranked lower by Selectical in later stages. Both of these are improvements at the starting end of the labeling. It seems Selectical is quite capable at identifying the first papers already relatively quickly, so improvements might be limited when enhancing the initial learning process. More gains could be made by optimizing the process of finding the last, harder relevant documents. Experimenting on a better curated larger and better curated query set may also improve performance or give insight in areas of improvement.

9 Conclusion

In this thesis we attempted to improve the performance of Selectical, a Continuous Active Learning model used for Technology Assisted Review used for screening prioritization, developed by Landscape. The goal is to find (nearly) all relevant documents with the least labeling workload. Our attempts mostly focused on alleviating the cold-start problem by using the Boolean query used to retrieve an initial set of documents as a more meaningful feature. Intuitively, the query describes what the researchers are looking for. We used Lexical models, as well as transformer-based rankers aiming to create relevance rankings based on Boolean queries using these Lexical rankers. We also attempted to use various transformer models to encode both queries and documents and proceeded to calculate the similarity between both to rank the documents. These rankings could be loaded in Selectical in order to find the first relevant documents faster, which should in theory enable the active learner to start learning earlier. During testing we found this gives an advantage in earlier stages of the labeling process, although this advantage is lost overtime due to the last papers being the hardest to find.

Lastly, we attempted to add a learning component to Selectical which has the similarity of the current document compared to previously labeled documents as a feature. This component was an attempt to use previously found relevant papers as seed documents to identify other relevant documents. This component had limited to no impact on the overall labeling load.

None of our attempts were very successful at improving Selectical as a model, however, some implementations had relatively good performance as stand-alone models for some topics. We therefore gained valuable insights and have determined some possible future steps to decrease labeling load. We suspect more information could be gained from queries. Having access to the original natural language query and research question might already increase performance. Letting the reviewer write a short text for what they are looking for could also work better in this case.

Systematic reviewing is a very complex task and the field of technology assisted review and natural language processing is evolving rapidly. New methods and models are developed at a rapid pace, so further possible improvements might become available in the near future.

References

[AD20] Ashwin Karthik Ambalavanan and Murthy V. Devarakonda. Using the contextual language model bert for multi-criteria classification of scientific articles. *Journal of Biomedical Informatics*, 112:103578, 2020.

[BRKF17] Wichor M. Bramer, Melissa L. Rethlefsen, Jos Kleijnen, and Oscar H. Franco. Optimal database combinations for literature searches in systematic reviews: a prospective exploratory study. *Systematic Reviews*, 6(1):245, Dec 2017.

[CDL16] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading, 2016.

[CG14] Gordon V. Cormack and Maura R. Grossman. Evaluation of Machine-Learning Protocols for Technology-Assisted Review in Electronic Discovery. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, page 153–162, New York, NY, USA, 2014. Association for Computing Machinery.

[CG15] Gordon V. Cormack and Maura R. Grossman. Autonomy and Reliability of Continuous Active Learning for Technology-Assisted Review, 2015.

[Cla18] Nigel S. Clarke. The basics of patent searching. *World Patent Information*, 54:S4–S10, 2018. Best of Search Matters.

[CM09] Gordon V. Cormack and Mona Mojdeh. Machine Learning for Information Retrieval: TREC 2009 Web, Relevance Feedback and Legal Tracks. In *Text Retrieval Conference*, 2009.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[DVRC17] Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. TREC Complex Answer Retrieval Overview. In *Proceedings of Text REtrieval Conference (TREC)*, 2017.

[FDC+23] Guglielmo Faggioli, Laura Dietz, Charles L. A. Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, and Henning Wachsmuth. Perspectives on large language models for relevance judgment. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '23, page 39–50, New York, NY, USA, 2023. Association for Computing Machinery.

[FS94] Edward A Fox and Joseph A Shaw. Combination of multiple searches. *NIST special publication SP*, 243, 1994.

[GC11] Maura R. Grossman and Gordon V. Cormack. Technology-Assisted Review in E-Discovery can be more effective and more efficient than exhaustive manual review. *Richmond Journal of Law and Technology*, 17:11, 2011.

[GC14] Maura R Grossman and Gordon V Cormack. Grossman-cormack glossary of technology-assisted review, the. *Fed. Cts. L. Rev.*, 7:85, 2014.

[GG13] S Gopalakrishnan and P Ganeshkumar. Systematic reviews and meta-analysis: Understanding the best evidence in primary healthcare. *J Family Med Prim Care*, 2(1):9–14, January 2013.

[HMVLB20] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020.

[HWKS12] Elke Hausner, Siw Waffenschmidt, Thomas Kaiser, and Michael Simon. Routine development of objectively derived search strategies. *Systematic reviews*, 1:1–10, 2012.

[Ker05] Anne Kershaw. Automated document review proves its reliability. *Digital Discovery and e-Evidence*, 5, 11 2005.

[KZ20] Omar Khattab and Matei Zaharia. 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*, pages 39–48, 2020.

[Lar99] Leah S Larkey. A patent search and classification system. In *Proceedings of the fourth ACM conference on Digital libraries*, pages 179–187, 1999.

[LB02] Edward Loper and Steven Bird. NLTK: the Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70, 2002.

[Liu09] Tie-Yan Liu. Learning to Rank for Information Retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[LMG08] Carol Lefebvre, Eric Manheimer, and Julie Glanville. Searching for studies. *Cochrane handbook for systematic reviews of interventions: Cochrane book series*, pages 95–150, 2008.

[LNY22] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. *Pretrained transformers for text ranking: Bert and beyond*. Springer Nature, 2022.

[LS18] Grace E Lee and Aixin Sun. Seed-driven document ranking for systematic reviews in evidence-based medicine. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 455–464, 2018.

[LYK+19] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, sep 2019.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[MMSO12] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR*, pages 60–63, 2012.

[MT20] Craig Macdonald and Nicola Tonellotto. Declarative experimentation ininformation retrieval using pyterrier. In *Proceedings of ICTIR 2020*, 2020.

[Mul94] Cynthia Mulrow. Systematic reviews: Rationale for systematic reviews. *BMJ (Clinical research ed.)*, 309:597–9, 10 1994.

[NJL20] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Document ranking with a pre-trained sequence-to-sequence model, 2020.

[NRS+17] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human-generated MAchine reading COmprehension dataset, 2017.

[NYCL19] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-Stage Document Ranking with BERT, 2019.

[OETM+15] Alison O'Mara-Eves, James Thomas, John McNaught, Makoto Miwa, and Sophia Ananiadou. Using text mining for study identification in systematic reviews: a systematic review of current approaches. *Systematic Reviews*, 4(1):5, Jan 2015.

[Ope22] OpenAI. ChatGPT, 2022.

[Ope23] OpenAI. GPT-4 Technical Report, 2023.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[RKO10] Herbert Roitblat, Anne Kershaw, and Patrick Oot. Document Categorization in Legal Electronic Discovery: Computer Classification vs. Manual Review. *JASIST*, 61:70–80, 01 2010.

[RSR+20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[SC22] Nima Sadri and Gordon V. Cormack. Continuous active learning using pretrained transformers, 2022.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

[SLZ18] Harrisen Scells, Daniel Locke, and Guido Zuccon. An information retrieval experiment framework for domain specific applications. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1281–1284, 2018.

[SYM+23] Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agent, 2023.

[SZK+17] Harrisen Scells, Guido Zuccon, Bevan Koopman, Anthony Deacon, Leif Azzopardi, and Shlomo Geva. A test collection for evaluating retrieval of studies for inclusion in systematic reviews. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1237–1240, New York, NY, USA, 2017. Association for Computing Machinery.

[SZK20] Harrisen Scells, Guido Zuccon, and Bevan Koopman. You can teach an old dog new tricks: Rank fusion applied to coordination level matching for ranking in systematic reviews. In Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva, and Flávio Martins, editors, *Advances in Information Retrieval*, pages 399–414, Cham, 2020. Springer International Publishing.

[SZK21] Harrisen Scells, Guido Zuccon, and Bevan Koopman. A comparison of automatic boolean query formulation for systematic reviews. *Information Retrieval Journal*, 24(1):3–28, Feb 2021.

[WMTO21] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. Pseudo-relevance feedback for multiple representation dense retrieval. In *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*. ACM, jul 2021.

[WSKZ22] Shuai Wang, Harrisen Scells, Bevan Koopman, and Guido Zuccon. Neural rankers for effective screening prioritisation in medical systematic review literature search. In *Proceedings of the 26th Australasian Document Computing Symposium*, pages 1–10, 2022.

[WSKZ23] Shuai Wang, Harrisen Scells, Bevan Koopman, and Guido Zuccon. Can chatgpt write a good boolean query for systematic review literature search?, 2023.

[WSMZ22] Shuai Wang, Harrisen Scells, Ahmed Mourad, and Guido Zuccon. Seed-driven document ranking for systematic reviews: A reproducibility study. In Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty, editors, *Advances in Information Retrieval*, pages 686–700, Cham, 2022. Springer International Publishing.

[ZCGS20] Haotian Zhang, Gordon V Cormack, Maura R Grossman, and Mark D Smucker. Evaluating sentence-level relevance feedback for high-recall information retrieval. *Information Retrieval Journal*, 23:1–26, 2020.

A Appendix

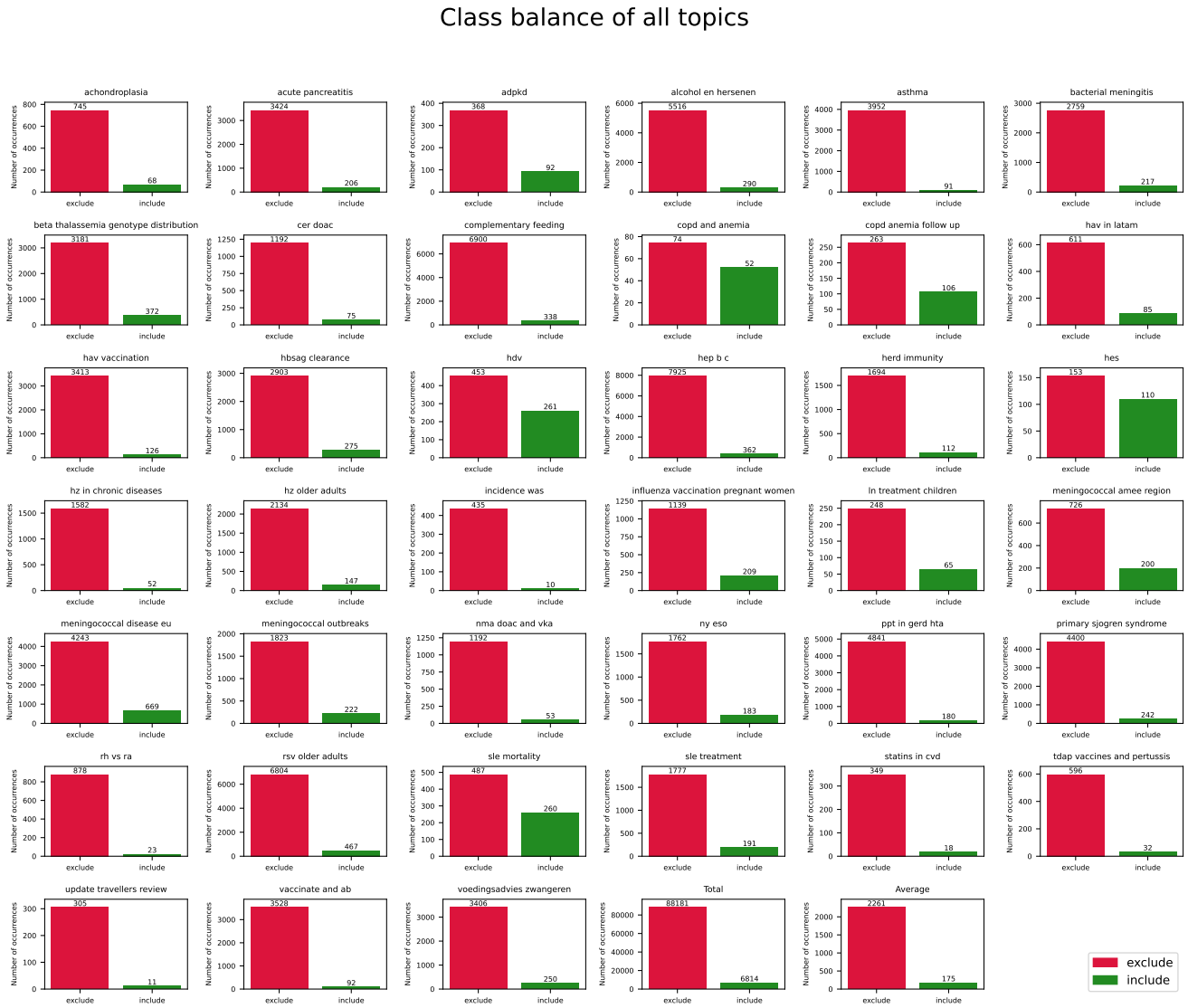


Figure 12: Full distribution of relevant and nonrelevant papers for each topic. Papers labeled as "include" and "doubt" are both visualised as "include".

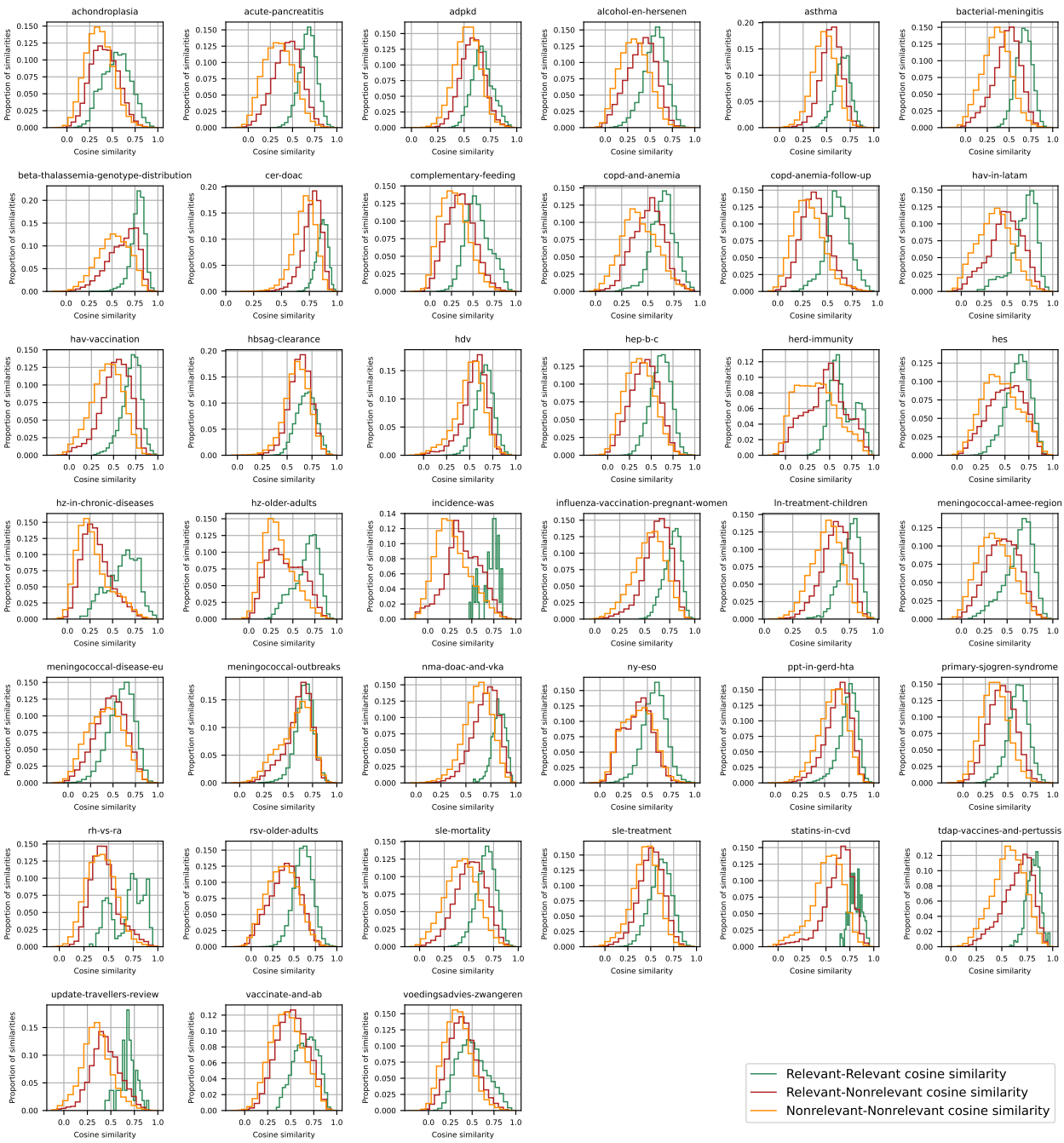


Figure 13: Cosine similarity distribution of all pairs of papers. For each topic, we calculated the cosine similarity between each paper and plotted the distribution of similarities for each possible relation: a relevant another relevant paper, a relevant and a nonrelevant paper, and a nonrelevant paper with another nonrelevant paper. One can see that the average similarity of relevant-relevant pairs often is slightly higher than the similarity of relevant-nonrelevant pairs. However, this is difference is not very pronounced in most studies. The distribution of similarities between nonrelevant-nonrelevant pairs is often wide.

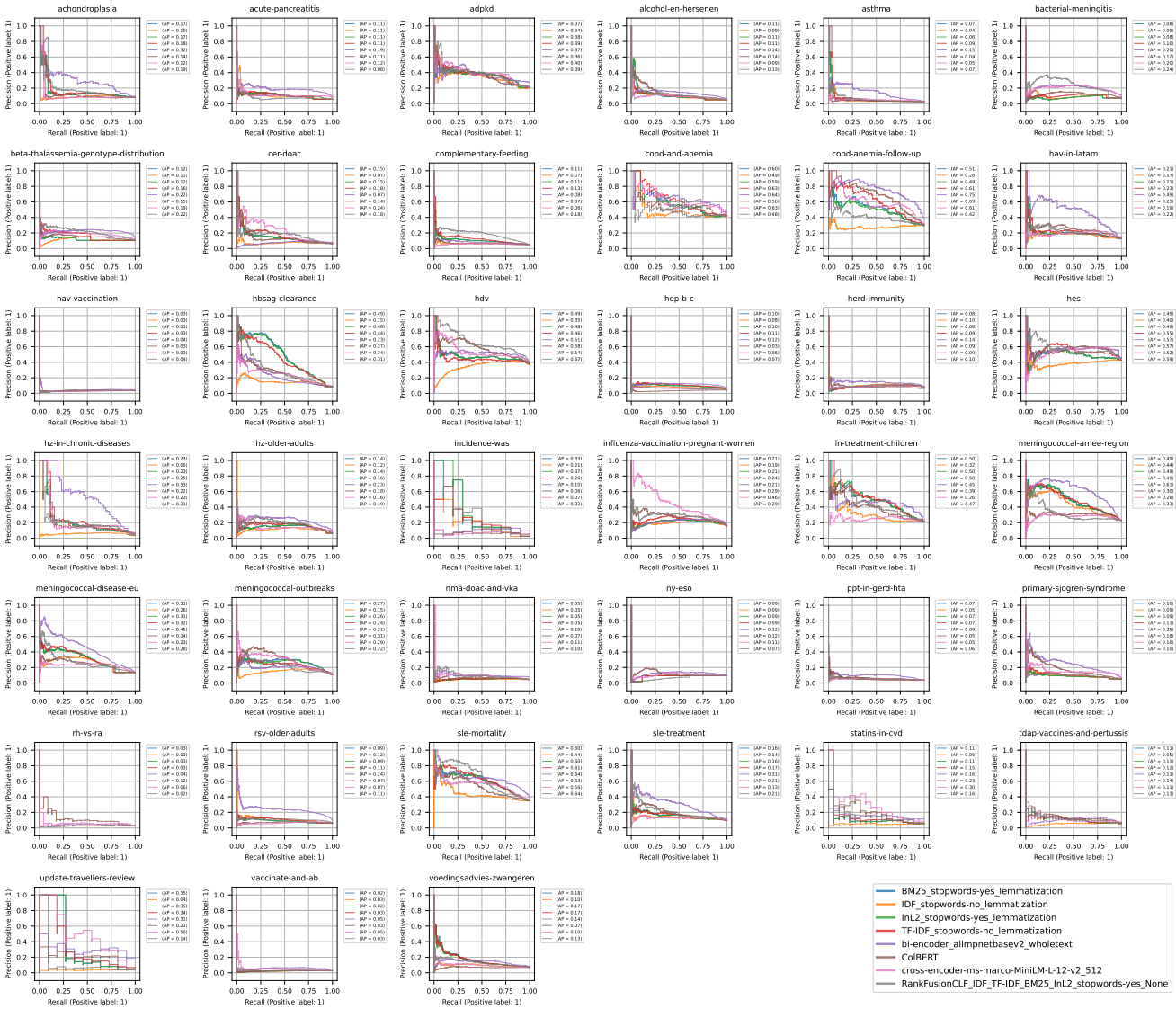


Figure 14: Precision-Recall curves per topic for each unsupervised model with the parameter configuration achieving the highest Mean Average Precision. The model performance vastly differs per topic. For a relatively high number of topics all models perform quite similarly. All models seem to fail in ranking some topics. Other topics have much better performance. In general, the bi-encoder performs the best of all models.

Study	no-init	init50	init100	init200	cos-no-init	cos-init100	cos-init200	cos-init50
achondroplasia	94.1%	93.4%	92.1%	97.9%	97.7%	95.2%	93.0%	97.3%
acute-pancreatitis	73.6%	66.8%	90.6%	79.9%	69.2%	86.4%	85.4%	81.7%
adpkd	94.4%	92.2%	90.3%	72.1%	93.1%	93.1%	72.7%	93.5%
alcohol-en-hersenen	94.8%	99.4%	97.5%	82.6%	90.1%	91.3%	97.9%	92.1%
asthma	88.2%	87.5%	64.0%	90.9%	98.4%	90.9%	75.4%	92.2%
bacterial-meningitis	82.3%	79.7%	81.8%	80.3%	81.6%	81.4%	80.1%	67.6%
beta-thalassemia	87.7%	82.5%	74.1%	72.6%	76.8%	85.4%	95.6%	83.6%
cer-doac	84.9%	87.9%	76.7%	89.2%	87.0%	94.4%	88.2%	93.1%
complementary-feeding	64.5%	52.4%	70.0%	72.8%	51.2%	51.3%	67.2%	66.5%
copd-and-anemia	98.4%	92.2%	85.9%	87.5%	98.4%	83.6%	87.5%	87.5%
copd-anemia-follow-up	71.2%	68.5%	75.5%	70.1%	77.1%	72.0%	71.7%	71.4%
hav-in-latam	98.7%	99.9%	91.4%	96.7%	69.0%	96.3%	89.0%	98.7%
hav-vaccination	90.5%	97.7%	98.4%	98.0%	98.9%	97.5%	98.6%	98.6%
hbsag-clearance	82.9%	94.6%	82.1%	77.8%	86.3%	85.9%	88.0%	79.7%
hdv	83.5%	94.6%	88.1%	98.2%	94.3%	97.2%	92.3%	94.4%
hep-b-c	72.8%	69.6%	75.8%	77.7%	58.1%	78.6%	66.0%	63.0%
herd-immunity	84.0%	87.4%	83.6%	84.4%	83.5%	72.1%	80.9%	95.8%
hes	76.6%	85.3%	92.1%	89.4%	79.6%	87.2%	87.5%	82.6%
hz-in-chronic-diseases	54.4%	68.8%	66.2%	57.1%	81.8%	77.9%	43.8%	43.6%
hz-older-adults	87.5%	73.5%	80.4%	79.3%	79.5%	70.9%	85.5%	92.9%
incidence-was	41.9%	34.3%	37.9%	24.0%	43.3%	41.0%	24.0%	48.7%
vacc.-pregnant-women	89.9%	99.1%	98.5%	91.4%	96.5%	94.8%	99.1%	91.8%
ln-treatment-children	91.4%	87.6%	71.1%	92.1%	85.1%	80.6%	87.9%	89.8%
meningococcal-amee-region	93.4%	95.3%	96.2%	81.2%	89.5%	81.8%	96.3%	88.6%
meningococcal-disease-eu	89.1%	74.0%	77.5%	76.7%	96.4%	93.2%	86.5%	84.1%
meningococcal-outbreaks	87.6%	95.0%	88.1%	93.3%	90.8%	87.2%	96.2%	96.4%
nma-doac-and-vka	93.1%	89.5%	60.0%	88.2%	92.1%	94.2%	60.4%	61.7%
ny-eso	77.9%	72.8%	69.3%	71.4%	81.0%	85.9%	65.4%	91.3%
ppt-in-gerd-hta	87.5%	72.9%	73.5%	79.9%	80.4%	91.5%	86.2%	92.4%
primary-sjogren-syndrome	84.8%	82.2%	71.4%	77.1%	70.9%	64.6%	73.3%	67.9%
rh-vs-ra	81.4%	75.9%	87.1%	70.5%	79.9%	91.4%	77.7%	58.9%
rsv-older-adults	54.4%	50.8%	56.0%	67.2%	53.4%	53.4%	46.1%	52.1%
sle-mortality	88.1%	87.8%	88.2%	88.6%	90.5%	86.2%	89.2%	83.3%
sle-treatment	91.5%	94.3%	91.9%	90.9%	86.5%	88.2%	85.2%	93.3%
statins-in-cvd	95.9%	94.3%	38.6%	42.7%	96.5%	35.3%	42.7%	92.4%
tdap-vaccines-and-pertussis	77.6%	81.7%	73.8%	76.5%	79.2%	76.6%	77.3%	76.6%
update-travellers-review	45.4%	42.9%	18.3%	18.3%	73.3%	18.3%	18.3%	37.9%
vaccinate-and-ab	70.6%	68.9%	88.2%	76.7%	59.7%	63.8%	55.7%	69.3%
voedingsadvies-zwangeren	84.8%	94.1%	93.1%	92.7%	90.8%	93.1%	94.7%	96.1%
average	81.8%	81.2%	77.8%	78.3%	81.7%	79.7%	77.1%	80.7%
weighted average	79.9%	77.6%	78.3%	79.4%	77.0%	79.2%	77.5%	78.7%

Table 17: Results in terms of labeling load of various setups using Selectical on all topics for reaching a recall of 100%. Lower numbers indicate a more effective ranker. Note that some dataset and all model names have been abbreviated. All no-init models did not use any model to initialize the active learner, and all init-models used the `allmpnetbasev2` bi-encoder, as in the other results. Columns containing "cos" denote the use of the cosine similarity active learner.

Study	no-init	init50	init100	init200	cos-no-init	cos-init100	cos-init200	cos-init50
achondroplasia	70.8%	90.5%	72.6%	59.0%	70.9%	92.0%	61.4%	64.5%
acute-pancreatitis	62.5%	55.1%	71.6%	57.3%	59.1%	49.6%	42.2%	53.3%
adpkd	74.2%	73.6%	73.8%	69.5%	77.3%	76.6%	68.8%	73.6%
alcohol-en-hersenen	82.5%	98.8%	88.6%	76.9%	82.3%	78.8%	78.3%	89.0%
asthma	85.9%	85.1%	62.5%	86.4%	97.7%	87.7%	69.2%	87.5%
bacterial-meningitis	74.3%	51.7%	74.4%	72.7%	75.7%	63.2%	68.6%	63.3%
beta-thalassemia	73.3%	64.4%	60.7%	60.1%	73.7%	73.0%	64.1%	72.5%
cer-doac	79.7%	80.0%	71.0%	84.7%	82.6%	75.3%	83.0%	73.7%
complementary-feeding	44.2%	33.6%	52.1%	44.2%	33.8%	38.0%	47.1%	46.8%
copd-and-anemia	96.1%	82.0%	82.0%	84.4%	78.1%	80.5%	84.4%	80.5%
copd-anemia-follow-up	70.6%	68.2%	73.0%	57.1%	71.4%	69.5%	55.8%	67.9%
hav-in-latam	95.4%	94.3%	85.7%	94.3%	67.1%	93.7%	87.5%	93.7%
hav-vaccination	82.3%	92.6%	91.9%	91.2%	94.3%	91.7%	95.3%	94.7%
hbsag-clearance	63.9%	64.9%	67.2%	67.1%	72.1%	74.6%	62.9%	65.6%
hdv	81.7%	82.3%	80.1%	83.8%	83.7%	83.8%	80.6%	85.8%
hep-b-c	56.9%	56.9%	59.9%	54.3%	47.5%	50.5%	51.4%	54.7%
herd-immunity	78.3%	83.9%	79.4%	82.2%	77.7%	70.8%	72.2%	90.6%
hes	74.7%	84.2%	90.9%	78.5%	76.6%	81.5%	80.4%	80.4%
hz-in-chronic-diseases	51.2%	26.2%	28.5%	27.9%	66.8%	48.0%	31.6%	31.0%
hz-older-adults	69.8%	71.5%	80.2%	77.6%	69.3%	68.1%	74.9%	56.6%
incidence-was	41.5%	28.7%	15.7%	15.7%	41.7%	15.7%	15.7%	39.0%
vacc.-pregnant-women	85.4%	95.0%	82.9%	81.8%	79.7%	81.9%	87.2%	79.6%
ln-treatment-children	88.9%	86.7%	68.6%	86.0%	84.4%	73.3%	87.6%	83.8%
meningococcal-amee-region	79.4%	91.3%	85.8%	76.7%	81.7%	68.8%	72.1%	75.2%
meningococcal-disease-eu	70.0%	69.3%	68.1%	68.2%	79.2%	75.1%	72.7%	69.7%
meningococcal-outbreaks	81.3%	85.8%	78.5%	88.4%	84.4%	85.4%	82.6%	84.4%
nma-doac-and-vka	73.4%	54.3%	57.8%	76.2%	55.1%	72.0%	57.2%	60.4%
ny-eso	64.1%	64.4%	66.3%	58.3%	61.7%	66.5%	58.8%	64.5%
ppt-in-gerd-hta	80.1%	54.4%	72.9%	63.8%	75.2%	88.0%	81.8%	82.8%
primary-sjogren-syndrome	74.7%	67.0%	65.9%	54.5%	68.8%	54.3%	59.3%	63.2%
rh-vs-ra	31.6%	29.7%	37.3%	40.9%	25.8%	38.4%	41.2%	39.0%
rsv-older-adults	43.6%	39.6%	46.1%	48.7%	43.5%	44.4%	36.1%	42.0%
sle-mortality	77.4%	75.4%	72.6%	72.5%	76.2%	73.1%	73.0%	77.0%
sle-treatment	88.6%	93.9%	88.1%	85.8%	82.6%	84.9%	84.8%	89.9%
statins-in-cvd	51.6%	41.6%	37.8%	40.5%	58.8%	31.0%	40.5%	43.5%
tdap-vaccines-and-pertussis	74.9%	80.0%	72.8%	55.6%	69.6%	64.2%	44.2%	73.0%
update-travellers-review	44.8%	11.0%	11.0%	11.0%	48.4%	11.0%	11.0%	11.0%
vaccinate-and-ab	64.3%	48.7%	46.5%	59.8%	54.9%	55.6%	54.9%	51.8%
voedingsadvies-zwangeren	79.0%	88.1%	88.2%	90.9%	86.4%	86.6%	88.4%	93.0%
average	70.8%	67.8%	66.9%	66.3%	69.6%	67.1%	64.3%	67.9%
weighted average	68.1%	64.9%	66.8%	65.3%	67.1%	65.9%	63.6%	67.0%

Table 18: Results in terms of labeling load of various setups using Selectical on all topics for reaching a recall of 99%. Lower numbers indicate a more effective ranker. Note that some dataset and all model names have been abbreviated. All no-init models did not use any model to initialize the active learner, and all init-models used the `allmpnetbasev2` bi-encoder, as in the other results. Columns containing "cos" denote the use of the cosine similarity active learner.

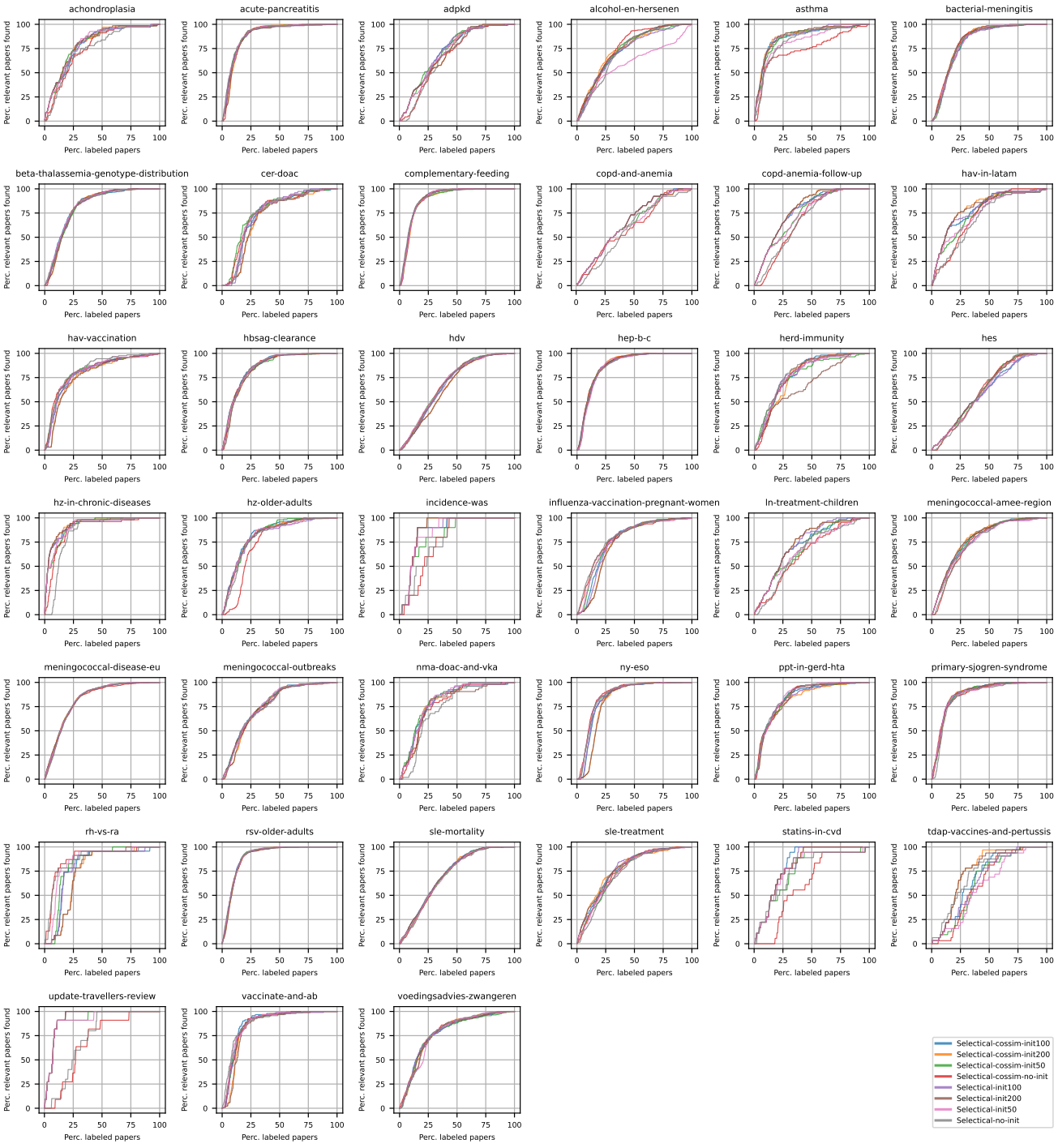


Figure 15: Progression of recall against number of documents that were labeled during simulation of all Selectical setups. It is noteworthy that almost every graph has a similar shape, where the first relevant documents are found relatively quickly and the last relevant documents are much harder to find. For some topics, it seems initializing using the bi-encoder can help but also hamper finding the first relevant documents. However, all setups converge near the end.

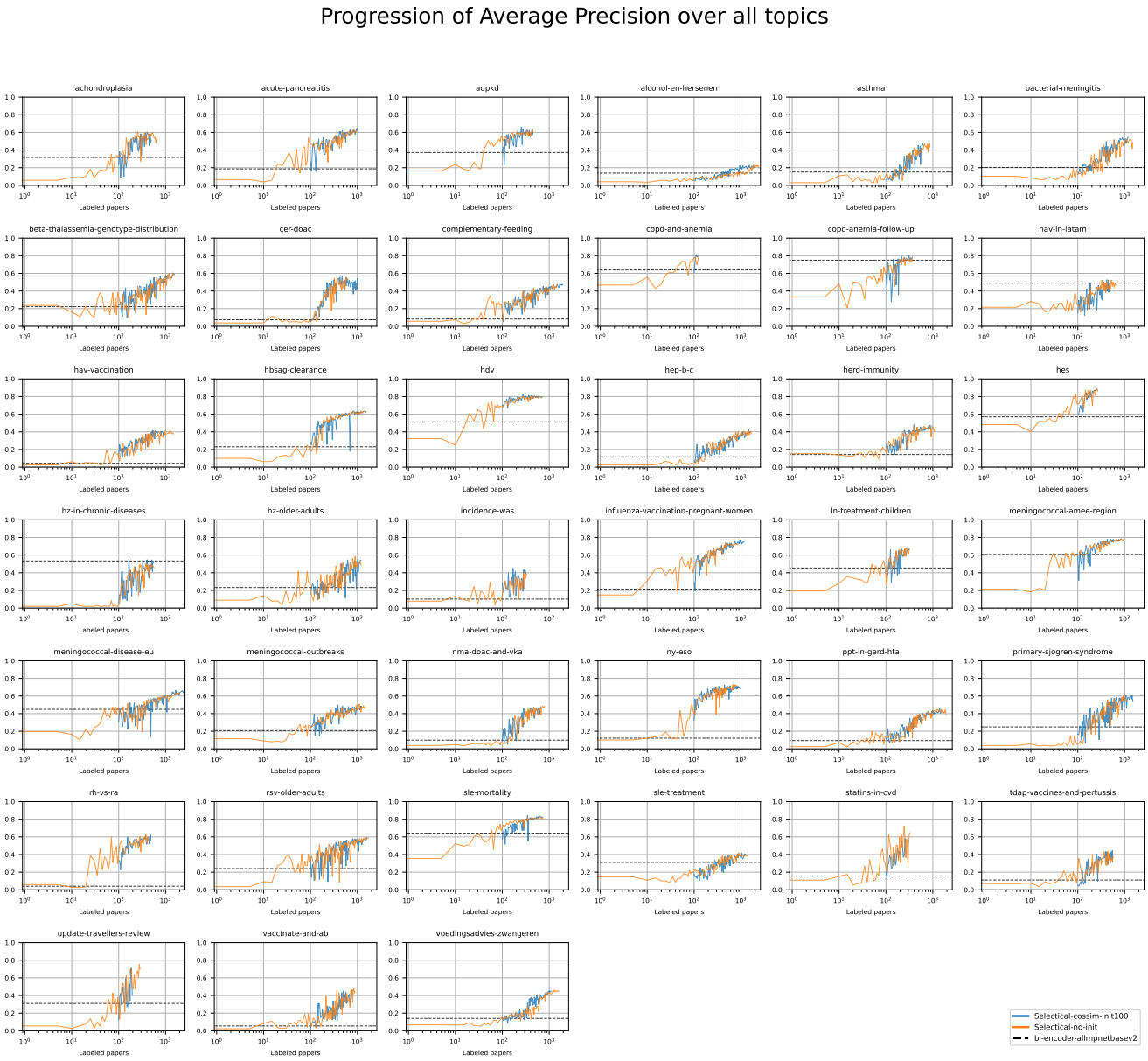


Figure 16: Progression of Average Precision per topic for "Selectical-cossim-init100" and "Selectical-no-init" (baseline) setup. The performance of the bi-encoder is also shown. As the bi-encoder does not learn from relevance feedback, the ranking and therefore Average Precision is constant. For most topics, Selectical will perform better than the bi-encoder after a few training sessions. However, for some topics, the bi-encoder performs much better and has a higher Average Precision for a long time, performing similarly to Selectical even in later training stages. Both Selectical setups converge to similar performance. The Average Precision is relatively inconsistent and has a high degree of variance per training step. This may be caused due to large changes in the model if a single new, relatively rare new relevant paper has been found.

All experiments in this thesis were performed on a server with the following specifications:

- CPU: AMD Ryzen 1950X 16-core with SMT
- RAM: 128GB DDR4
- GPU: Nvidia GTX 1080ti with 11GB GDDR5X VRAM
- Storage: Sufficiently fast NVMe SSD storage