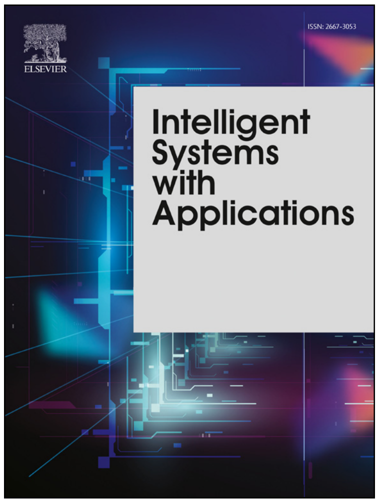


Transferring knowledge between topics in systematic reviews

Alessio Molinari and Evangelos Kanoulas

PII: S2667-3053(22)00087-4
DOI: https://doi.org/10.1016/j.iswa.2022.200150
Reference: ISWA 200150

To appear in: *Intelligent Systems with Applications*

Received date: 19 June 2022
Revised date: 25 October 2022
Accepted date: 4 November 2022

Transferring knowledge between topics in Systematic Reviews

Alessio Molinari[a], Evangelos Kanoulas[b]

[a] *Institute of Information Science and Technologies "Alessandro Faedo" - ISTI, National Research Council (CNR), Pisa, Italy*
[b] *University of Amsterdam, Amsterdam, NL*

Abstract

In the medical domain, a systematic review (SR) is a well-structured process aimed to review all available literature on a research question. This is however a laborious task, both in terms of money and time. As such, the automation of a SR with the aid of technology has received interest in several research communities, among which the Information Retrieval community. In this work, we experiment on the possibility of leveraging previously conducted systematic reviews to train a classifier/ranker which is later applied to a new SR. We also investigate on the possibility of pre-training Deep Learning models and eventually tuning them in an Active Learning process. Our results show that the pre-training of these models deliver a good zero-shot (i.e., with no fine-tuning) ranking, achieving an improvement of 79% for the MAP metric, with respect to a standard classifier trained on few in-domain documents. However, the pre-trained deep learning algorithms fail to deliver consistent results when continuously trained in an Active Learning scenario: our analysis shows that using smaller sized models and employing adapter modules might enable an effective active learning training.

Keywords: Technology-Assisted Review, Systematic Reviews, Transfer Learning, Machine Learning, Deep Learning,

1. Introduction

A systematic review (SR) is a well-structured process that allows scientists to exhaustively and without bias review all available literature on a research question and it constitutes the cornerstone of evidence-based medicine. Conducting an SR however is a laborious and expensive task [1, 2, 3], which takes on average 1.72 years to complete [2], jeopardising the quality of healthcare. Automating a systematic review with the use of technology has received the interest of the Information Retrieval community, among others, which contributes

Email addresses: alessio.molinari@phd.unipi.it (Alessio Molinari), e.kanoulas@uva.nl (Evangelos Kanoulas)

to the Technology-Assisted Review (TAR) by enhancing the discovery of articles to include in the review.

Research effort has been mostly directed towards the query formulation stage of an SR to retrieve the initial set of articles [4, 5] or the priority screening stage to re-rank retrieved articles [6, 7, 8]. Datasets are now freely available[1] to further study the problem. In this work, we focus on the priority screening stage, i.e., we assume to have a set of unlabeled documents D as a result of a query issued on one or more dedicated scientific article databases and the goal is to prioritize them for review.

Regarding this second stage, much work has focused on improving the relevance feedback (aka active/online learning) process [6] that allows to continuously re-rank articles while reviewing, or on finding a good stopping criterion [9, 10, 11] to stop reviewing. However, to the best of our knowledge, little effort has been made towards transferring knowledge from previous and unrelated SR topics to the current one [12, 13]. In this work we want to explore the possibility of using previous Systematic Reviews in order to pre-train a model, which is later used for a zero/few shot classification or ranking of a new, unseen, review topic. As a matter of fact, current state-of-the-art TAR frameworks do not leverage previous systematic reviews: the motivation behind this work is then that of exploring whether using past reviews can be more effective than doing no pre-training at all. More specifically, we investigate two research questions:

RQ1. Can we transfer the knowledge acquired on previous systematic reviews, and if so, to which extent?

RQ2. Can we keep training our pre-trained models in the active learning process?

The contributions of this paper are:

1. An extensive analysis and comparison between transformer-based deep learning models in a zero-shot setup versus traditional machine learning models with a few-shot setup, demonstrating the advantages of out-of-domain pre-training;
2. A method to jump-start the active learning process from the zero-shot ranking obtained from our models that improves the document retrieval, albeit not consistently;
3. An extensive analysis on continual training via active learning of deep ranking models, that inform researchers and practitioners on which scenarios and techniques might enable the training of these models in an active learning process (Section 6.3).

1.1. TAR for systematic reviews: an overview

In evidence-based medicine, a systematic review is conducted by a physician in several stages. Given a research question:

[1]See for instance `https://github.com/CLEF-TAR/tar`

1. The physician prepares a query which is issued on one or more search engines (for medical literature);
2. An initial pool of documents D (based on the search engine ranking) is retrieved;
3. The physician reads (i.e., reviews) D's abstracts. Only the subset of documents D_r which are deemed $\underline{r}$elevant is kept;
4. The D_r documents are now read and reviewed in their entirety. Again, we filter out all the non-relevant documents;
5. Finally, the remaining documents will then form the set of documents included in the systematic review.

TAR algorithms can assist the physician in step 3 and 4; however, in this work we will focus exclusively on step 3. More specifically, the TAR process is structured in the following manner:

1. An initial set of labelled documents S is provided. This set should contain at least one positive (and it often consists exclusively of this single positive) document;
2. A machine learning algorithm is trained on S and outputs scores (or probabilities) on the remaining $D \setminus S$ documents;
3. An active learning policy (e.g. Continuous Active Learning, see Section 2) decides, based on these scores, which and how many documents to show to the reviewer (i.e., the physician);
4. The reviewer reads the selected documents abstracts, deciding whether they are relevant or not;
5. The seed set S is augmented with the new labelled documents and the process starts again, until a review budget is exhausted or a stopping rule condition is met[2].

We give a graphical representation of the process in Figure 1, and a more detailed algorithm in Algorithm 1.

2. Related Work

Technology-Assisted Review (TAR) frameworks for systematic reviews have received an increasing interest by the IR community in recent years. Focusing on the article retrieval, prioritization and reviewing stage of the SRs, one of the most well-known state of the art approaches is the Continuous Active Learning (CAL) algorithm [9]. The algorithm starts from a seed positive document, trains a machine learning classifier (logistic regression or SVM) and iteratively ranks an increasing number of documents; at every iteration, it receives assessments on the top ranked documents and re-trains the model.

[2]In this work, we do not focus on budget and/or stopping rules. This means in our experiments in Section 6 we review the whole dataset.

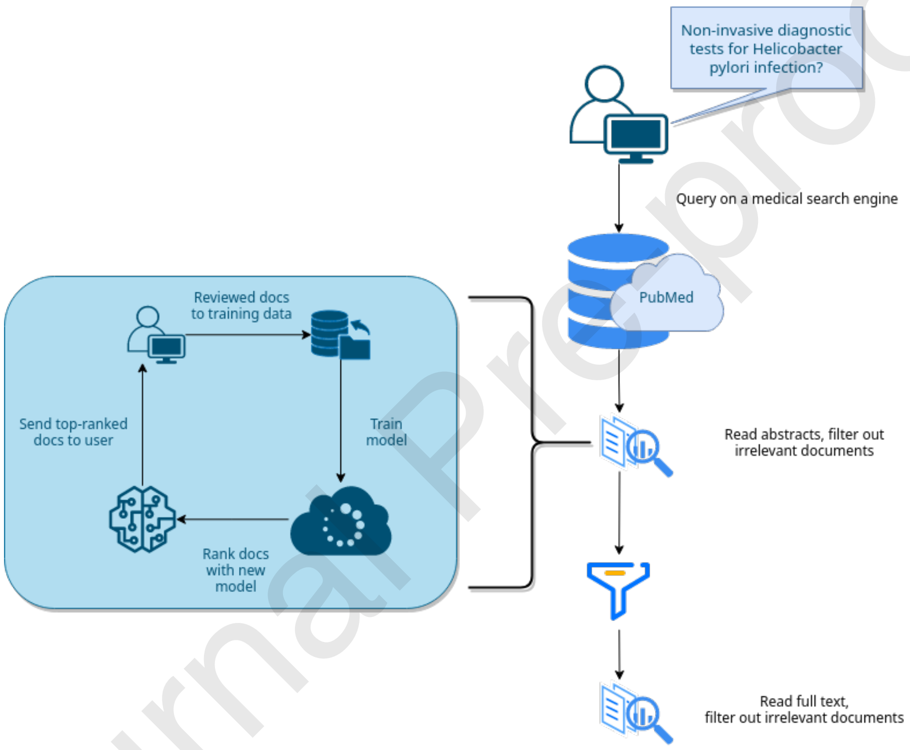

Non-invasive diagnostic tests for Helicobacter pylori infection?
Query on a medical search engine
PubMed
Read abstracts, filter out irrelevant documents
Read full text, filter out irrelevant documents
Reviewed docs to training data
Train model
Rank docs with new model
Send top-ranked docs to user

Figure 1: Diagram of a technology-assisted systematic review process.

Algorithm 1: TAR process for a systematic review

```
Input: Pool of documents $D$ to be reviewed; Active learning policy $a$;
       Initial seed set $S$; Batch size $b$; Budget $t$; threshold value $\alpha$;
Output: Relevant labelled documents $D_r$
$i \leftarrow 0$;
$L \leftarrow S$;
$\phi_i \leftarrow$ train_clf($L$);
/* We review a batch $B_i$ of documents and add it to the labelled
   (training) set $L$                                              */
$B_i \leftarrow$ select_via_policy($\phi_i, a, D, b$);
$L \leftarrow L \cup B_i$;
while $|L| < t$ do
    $i \leftarrow i + 1$;
    $U \leftarrow D \setminus L$;
    $\phi_i \leftarrow$ train_clf($L$);

    $B_i \leftarrow$ select_via_policy($\phi_i, a, D, b$);
    $L \leftarrow L \cup B_i$;
end

// Gather the relevant documents
$D_r \leftarrow \langle\rangle$;
foreach $d \in D_r$ do
    // $y_d$ is a binary variable which equals 1 when $d$ is relevant
    if $y_d = 1$ then
        append($D_r, d$);
    end
end
return $D_r$
```

Regarding the usage of pre-existing systematic reviews to pretrain a model which is later applied to a new SR topic, an interesting work (as well as the first one attempting such an experiment, to the best of our knowledge) is [12]. In this study, the authors train an SVM algorithm on 23 previously conducted SRs and later apply it to one unseen topic, using a Leave-One-Out setup; the experimental results show how such an approach is able to consistently deliver better results than the baseline. That said, they assume the availability of topic-specific training data, which we do not in this work (shifting the context to a real zero-shot scenario).

A more recent work where a transfer learning approach is leveraged is [13]. Here the pre-training of a model on previous SRs is part of a larger framework, which deals with all the stages of a systematic review. The experiments conducted are focused on the whole pipeline rather than the effectiveness of the training procedure. Nonetheless, their results show that pre-training can indeed be useful and that knowledge can be transferred between different topics. Pickens and Gricks [14] conduct several experiments to measure whether "portable" models can be trusted (and whether they are effective) in TAR applications; however, their experimentation mostly focuses on in-topic training (i.e., the training set (source) is coming from the same distribution of the test (target) set).

Most of these approaches experiment with machine learning algorithms such as SVM or logistic regression: Deep Learning (DL) models are usually not considered fit for such a task, as they depend on too many parameters (and data is usually scarce) and are easily outperformed by classic ML algorithms [15]. That said, there has been some work on testing DL models for TAR applications: Yang et al. [15] tried to use a just-right fine-tuned BERT [16] in the active learning process for e-Discovery. BERT is first fine-tuned with the masked language task on the available documents, and later continuously trained for a fixed number of epochs during the active learning review process. Their results show how in some cases BERT can actually achieve slightly better results than linear models, but postpone further experimentation to future works. Similarly, Zhao et al. [17] explored whether several models (BERT, logistic regression) trained (or fine-tuned) on a given training set can perform well on new data not seen during training: they showed how pre-training can usually bring to good performances, but the results are not consistent across all tested datasets (i.e., the pre-trained models can completely fail to transfer knowledge in some cases).

While the different learning architectures that we test (see Section 4.1) are first trained/fine-tuned on a set of previous systematic reviews topics for which we have the reviewers' final decisions, they are later tested on a previously unseen SR topic. We assume not to have any kind of training data for this latter test topic; the reviewer's opinion is elicited via an active learning approach. As previously mentioned, Cormack and Grossman's Continuous Active Learning (CAL) algorithm [9] is the de-facto standard in TAR applications (both for e-discovery and systematic reviews). Hence, we use the CAL algorithm as the main and only active learning approach to emulate data annotation in our experiments. As we will see in more details in Section 6, the CAL algorithm will be both used as a baseline (using a classical logistic regression as its classifier) and as the active learning methodology to continuously train our transfer-learning models. Regarding its implementation, we follow the configuration described in [9] (unless differently stated), where we use an exponentially increasing batch size b, starting from $b = 1$, and using the title of the topic as the seed (and only known) positive document.

3. The Dataset

For our experiments, we use the 2019 CLEF-TAR collection[3]. We keep the training/testing splitting as given in the GitHub repository for Task 2. More precisely, we have 53 topics (i.e., systematic reviews) for training and 8 topics for testing. However, one of the testing topics, more specifically topic "CD011686" is also present in the training sets, and as a result we remove it from our test set, reducing it to 7 topics. In order to retrieve documents' abstracts we used the HTTP API available at `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi`, downloading abstracts via PUBMED[4]. Due to issues with the API or the availability of the documents, we were able to retrieve most but not all of the documents. For each topic in the training data, the minimum amount of documents we have is 65 (123 for testing data), whereas the maximum is 79, 783 (9, 730 for testing data). On average, we have a prevalence of the positive class of 0.04 (0.07 for testing data): that is, the dataset is strongly unbalanced in favor of the negative class, as it is often the case in TAR tasks. We present more in-detail information about the test topics in Table 1. Moreover, we show a few examples from our dataset in Table 2 and 3 (for training and testing topics, respectively).

4. Methodology and experimental design

4.1. Learning algorithms

With the goal of understanding whether we can transfer information between different systematic review topics, we explore and employ different learning architectures:

- a classical **Logistic Regression (LR)** classifier, which is one of the well-established standard learning models used in TAR for systematic reviews literature;
- the **BioBERT** architecture [18], a BERT [16] based model specifically trained on scientific and medical data which could thus be expected to achieve good performances on our task. We fine-tune it with a pairwise loss;

[3]Available at `https://github.com/CLEF-TAR/tar/tree/master/2019-TAR`
[4]`https://pubmed.ncbi.nlm.nih.gov/`

Topic	Size	# Positives	Prevalence
CD012567	6736	12	0.0018
CD012768	132	46	0.3484
~~CD011686~~	~~9730~~	~~65~~	~~0.0067~~
CD012080	6644	78	0.0117
CD012669	1261	72	0.0571
CD012233	473	44	0.0930
CD008874	2383	119	0.0499
CD009044	3170	12	0.0038
Avg.	3816.12	56	0.07155

Table 1: Size, number of positives and prevalence of the positive class for the test dataset. Topic "CD011686" was originally present in the 2019 CLEF-TAR testing collection, but was removed since it is also present among the training topics.

Training data	
RQ	Abstract
Rapid diagnostic tests for diagnosing uncomplicated non-falciparum or Plasmodium vivax malaria in endemic countries	To determine the competence of community health workers (CHWs) to correctly assess, classify and treat malaria and pneumonia among under-five children after training [...]
Combination of the non-invasive tests for the diagnosis of endometriosis	It has been suggested that histologic subtype of ovarian cancer is a factor that determines the chemoresponsiveness of tumor. In this study, we wanted to clarify the prognostic significance of histologic subtype and its correlation to expression of chemoresistance-related proteins (CRPs) in ovarian cancer. [...]
Molecular assays for the diagnosis of sepsis in neonates	One of the leading causes of severe childhood gastroenteritis are group A rotaviruses, and they have been found to be associated with ∼40% of the annual gastroenteritis-associated hospitalizations in young Danish children ¡ 5 years of age (Fischer et al., 2011) [...]

Table 2: Three samples from our training data. We can see the research question (i.e. what we are looking for) on the left, and parts of a random picked abstract on the right.

Testing data	
RQ	Abstract
Non-invasive diagnostic tests for Helicobacter pylori infection	To describe and explore the natural history of Helicobacter pylori infection and chronic gastritis in terms of gastric mucosal atrophy and ulcer development over time in a population-based cohort. [...]
Point-of-care ultrasonography for diagnosing thoracoabdominal injuries in patients with blunt trauma	Bedside lung ultrasound (LUS) is useful in detecting radio-occult pleural-pulmonary lesions. The aim of our study is to compare the value of LUS with other conventional routine diagnostic tools in the emergency department (ED) evaluation of patients with pleuritic pain and silent chest radiography (CXR). [...]
Triage tools for detecting cervical spine injury in pediatric trauma patients	We reviewed published radiographic and cadaver series describing the incidence of the anatomical anomaly ponticulus posticus and discuss its relevance to C1 lateral mass screw (C1LMS) insertion. [...]

Table 3: Three samples from our testing data. We can see the research question (i.e. what we are looking for) on the left, and parts of a random picked abstract on the right.

- a deep learning model based on the transformer architecture [19] where the self-attention mechanism is actually computed between documents and not between tokens. This model was first proposed in [20]; we refer the reader to the original work for a more thorough and in-depth explanation of this model. We test the model by maximizing the NDCG metric, more specifically we implement the deterministic NeuralNDCG [21]. We call this model the **DL Ranker or Ranker**;
- we also test the same architecture with a cross-entropy loss; we call this model the **DL Classifier or Classifier**;

The reasons behind our choice to test another deep learning architecture other than BioBERT are strongly tied to the learning setting we are confronted with in systematic reviews: firstly, despite having a decently sized training set when we merge together past systematic reviews, the testing and fine-tuning of our models is done on single topics, whose sizes may be very small (i.e., few hundreds of documents, with a very low positive prevalence); we believe that having less parameters to learn might ease the learning/fine-tuning of models when continuously trained in an active learning process. The second reason that motivates us in using the ranker model of [20] is that we would also like to compare the more traditional classification and/or pairwise approaches (i.e., the logistic regression and BioBERT) to a list-wise capable architecture.

4.2. Data Preprocessing

As explained in Section 4.1 we experiment with different architectures, which require different data representation and organization:

- The Logistic Regression, the DL Ranker and the DL Classifier are trained with the same data representation. We see this in detail in Section 4.2.1;
- We train the BioBERT model with a pairwise loss, where the aim of the model is that of correctly prioritizing one of the documents in the pair. For this reason we merge together pairs of documents as explained in Section 4.2.2;
- Since we compare with the CAL baseline (as implemented in [9]), we also need a representation suited to its logistic regression; we simply default to a standard TF-IDF representation.

We will now look more in details to the different techniques we use to preprocess our data before feeding it to the models.

4.2.1. Document embeddings and list-wise structure

For the LR, the DL Ranker and DL Classifier models, we preprocess data by tokenizing each document and transforming it into a fixed-dimensional vector of features. The title of the topic (i.e., the research question of the SR) is prepended to the text of each document. More precisely, for each document we have a feature vector v of size $E = 768$. This vector v is the average of the non-finetuned BioBERT embeddings for the tokens of a document $d \in D$.

The architecture used for the DL Ranker and Classifier requires a matrix $s \times E$ of document embeddings as input (where s stands for sequence). Ideally, s should be equal to the number of documents we need to rank (i.e., $|D^{\{\text{tr},\text{te}\}}|$, where with overscript "tr" or "te" we indicate the training or test split of D) but the computational costs would be unsustainable when we have thousands of documents to rank. We have thus to pick a value of $s < |D^{\{\text{tr},\text{te}\}}|$. Due to hardware constraints, we choose $s = 1000$ in our experiments. Moreover, we also fix a batch size $b = 512$ such that the DL Ranker and Classifier are finally fed a $b \times s \times E$ matrix as input, and return a $b \times s \times o$ matrix as output, where o is the number of neurons in the output layer (this is always 2 for the Classifier and we set it at 100 for the Ranker, see also [20, 21] for the original implementation). For the DL Ranker, we then take the average on the third dimension, such that we have a matrix $b \times s$ of scores for each document. For the DL Classifier, we also "augment" the training set D^{tr} with a pre-defined number of documents randomly sampled from other topics (we sample a number equal to 20% of $|D^{\text{tr}}|$ size): this should help the classifier to generalize better over the different topics; clearly, this cannot be done with the DL Ranker, as it would not make sense to rank documents for a topic t_i higher than a topic t_j or viceversa.

Finally, each sequence s is built by randomly sampling s examples from the dataset without replacement. Notice that since we only elaborate a sequence s of documents for every batch, when applying the model to the test topics we first classify/rank all batches and then aggregate the scores together to obtain the ranking/classification output for each $d^{\text{te}} \in D^{\text{te}}$.

4.2.2. Pairwise document representation

We use this pairwise representation for the BioBERT fine-tuning. For n times, we randomly pick a relevant and an irrelevant example and we combine them together: a "new" document is then formed, where we have the title of the SR topic separated by a [SEP] token from the text of the first document, which is in turn separated by another [SEP] token from the text of the second document. Whether the relevant document is first or last is decided by a fair coin flip. Since BioBERT needs many data points to be fine-tuned, for any D^{tr}, we set $n = 1.2 \cdot |D^{\text{tr}}|$, creating a training set which is 20% larger than the original size of D^{tr}.

4.2.3. TF-IDF

The classical TF-IDF representation is only used for the logistic regression trained in the CAL algorithm used as a baseline. This representation is built using the TFIDFVECTORIZER class exposed by SCIKIT-LEARN[5] API.

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

4.3. Rankings and evaluation measures

As explained in Section 2, we use the CAL algorithm as a baseline in our experiments. We use a standard non-pretrained logistic regression (which we call **NP Logistic**) as the CAL's learner. Notice that, clearly, comparing CAL's performances with the zero-shot rankings is not a fair comparison since the NP Logistic has access to and is continuously trained on in-topic data.

Furthermore, throughout our experiments, we will report metrics and results on two different types of ranking:

- A ranking which is the output of a model (be it a set of probabilities or a vector of scores) on the set of all documents to be ranked. We call this **full reranking**;
- The ordering of the documents resulting from the CAL process, that we call the **CAL ordering**. More precisely: at every iteration of CAL, we take the top k documents and have a reviewer annotating them. What we call CAL ordering is then the order in which the reviewer annotates the documents throughout the process.

4.3.1. Evaluation metrics

In order to evaluate the models rankings or the CAL ordering, we use two of the most well-known metrics in TAR literature: Mean Average Precision (MAP) and Work Saved Over Sampling (WSS). Average Precision (AP) is computed as:

$$\text{AP} = \frac{1}{rel}\sum_{j}\text{Precision}(j), \tag{1}$$

where rel is the number of relevant documents and Precision(j) is the precision at the jth item. We take the average of this metric over all the testing topics, calling it MAP.

On the other hand, with WSS we want to measure how beneficial is our ranking with respect to a random ordering of the documents. More precisely, WSS at a threshold t is computed as:

$$\text{WSS@t} = (\text{TN} + \text{FN})/N(1 - t) \tag{2}$$

In our experiments we show WSS at the 85%, 95% and 100% thresholds (WSS@{85, 95, 100}%).

5. Implementation details

The aim of our experimentation is to answer research questions RQ1 and RQ2 (see Section 1). Unless otherwise stated, we train our DL Ranker and DL Classifier models for 500 epochs, using the Adam optimizer [22]. BioBERT[6]

[6] The pre-trained model is downloaded from the HuggingFace Hub, available at `https://huggingface.co/models`

is instead fine-tuned with a classification head with two output neurons for a maximum of 10 epochs. However, we employ a typical early stopping strategy with patience on the loss set to 10 update steps, which in practice usually stops the training set before reaching 10 epochs. The DL models (except for BioBERT) are implemented using the PyTorch Python library[7], whereas for the Logistic Regression we use the standard scikit-learn library[8] implementation (code will be made available in the near future).

As anticipated, for the DL Ranker and Classifier we experiment with two different losses: we use (i) a Cross-Entropy loss for the DL Classifier model; given the extremely unbalanced datasets (see Section 3), we also use fixed class weights of 0.2 and 0.8 for the negative and positive class respectively. Furthermore, for all of our experiments with the DL models, we use only one transformer encoder layer; we leave experimentation with different number of encoders and attention heads to future works. (ii) For the ranking loss function, we directly maximize an approximation of the NDCG metric, more precisely the deterministic Neural-NDCG [21]. While we refer the reader to [21] for implementation details, we notice that in order to approximate the permutation matrix that would sort the input, we have a temperature parameter τ which we can use to control the degree of approximation (when $\tau = 0$ we get the exact permutation matrix): we found this parameter to be highly susceptible to the size of the data fed to the model, but we postpone any precise analysis on this to future works.

For the fine-tuning of the BioBERT model, we use a pairwise loss function usually known as the Margin (or Margin Ranking) loss. That is, given two scores x_i and x_j, and a label $y = 1$ when x_i should be higher than x_j and $y = -1$ viceversa, the loss is computed as:

$$L(x_i, x_j, y) = \max(0, -y \cdot (x_i - x_j) + m) \tag{3}$$

where m is the margin, which we actually set to 0, following the PyTorch implementation default[9] as of version 1.10.1.

All of our experiments are run on a maximum of two NVIDIA Tesla T4 GPU (with 16GB of RAM each) on a multi-processor machine (kindly made available by the Computer Science department of the University of Pisa). Training times are not particularly demanding, as the BioBERT fine-tuning took around 36 hours, whereas 20 minutes were enough for the DL models and as little as 28 seconds for the logistic regression.

6. Results

We show in this section the experiment results both for our first and second research questions (RQ1 and RQ2, see Section 1). We also present the results of a hyperparameter search for the DL models (Section 6.3): this last section

[7]`https://pytorch.org/`

[8]`https://scikit-learn.org/stable/`

[9]`https://pytorch.org/docs/stable/generated/torch.nn.MarginRankingLoss.html`

	CAL ordering	NP Logistic (no pre-training)	Zero-shot rankings			
			DL Rank.	DL Class.	BioBERT	LR
MAP	**0.240** ±0.178	0.126 ± 0.132	0.211 ± 0.168	0.185 ± 0.163	<u>0.226</u> ±0.162	0.219 ± 0.166

Table 4: MAP for the zero-shot rankings. We show the CAL ordering and CAL's NP Logistic full reranking after 10 documents have been annotated for comparison. Best result overall is in **bold**, whereas the best result among the zero-shot rankings is <u>underlined</u>.

should serve as a basis for further research, in order to understand how and where future works might focus to successfully continuously train DL models in active learning scenarios.

6.1. RQ1: Can we transfer knowledge?

In order to answer this first question, we train our models on our training topics and apply them to the testing topics without any further training, to see whether we can actually obtain a good zero-shot ranking of the documents.

Evaluating the zero-shot ranking. As stated in Section 4.3, we compare our results (full reranking) with the document ordering coming from Cormack's CAL classical implementation (what we called the CAL ordering). We show the Mean Average Precision (MAP) of the zero-shot rankings in Table 4; we also show the MAP for the CAL ordering and the CAL's NP Logistic ranking after 10 documents have been annotated (which we might call a few-shots NP Logistic). Notice how all the zero-shot models achieve rather good performances, obtaining a better MAP than the few-shots NP Logistic. BioBERT seems to be the best model, closely followed by the Logistic Regression (LR). As expected, the CAL baseline is able to achieve a stronger MAP than the zero-shot rankings' (since its logistic regression is being trained on in-topic data). Nonetheless, this shows that the pre-trained models are able to successfully transfer knowledge between topics.

Jump-starting the CAL algorithm. To test if our zero-shot rankings are beneficial to the reviewing process (i.e., if we can achieve a higher recall earlier), we propose to jump-start the CAL algorithm from the top-10 documents coming from our zero-shot rankings. With "jump-starting" CAL we mean:

1. we pre-train a model on the training topics;
2. we rank the current new (and unseen) topic and take the model top-10 documents;
3. we obtain the labels for these 10 documents;
4. we train CAL's logistic regression on these 10 documents (using TF-IDF features) and start the CAL algorithm from there, following [9] thereafter.

Notice that, for some topics, the pre-trained models failed to retrieve any positive instance in the top-10 documents: the DL Classifier failed on 3 topics out of 7, of which the DL Ranker failed on 2 and BioBERT and the LR failed on

	NP Logistic	DL Rank.	DL Class.	BioBERT	LR
R@10	0.012 ± 0.017	0.028 ± 0.022	0.029 ± 0.040	**0.069** ± 0.055	0.046 ± 0.032

Table 5: Recall@10 (R@10) for the different pre-trained models and the NP Logistic baseline after annotating 10 documents. Notice the recall is measured on what we called the full reranking and not on the CAL ordering. All the pre-trained zero-shot models obtain a higher Recall@10.

	CAL ordering				
	NP Logistic	DL Rank.	DL Class.	BioBERT	LR
WSS@85%	0.494 ± 0.241	0.501 ± 0.245	0.499 ± 0.243	0.501 ± 0.243	**0.508** ± 0.254
WSS@95%	0.475 ± 0.304	0.466 ± 0.205	**0.480** ± 0.159	0.457 ± 0.328	0.460 ± 0.334
WSS@100%	0.372 ± 0.306	0.371 ± 0.297	0.373 ± 0.304	0.369 ± 0.307	**0.378** ± 0.304
MAP	0.357 ± 0.134	0.378 ± 0.151	0.387 ± 0.136	**0.468** ± 0.124	0.399 ± 0.133

Table 6: WSS@{85, 95, 100}% and MAP for the jump-started CAL. The NP Logistic is the classical CAL implementation, starting from a seed document. The other columns indicate from which ranking we take the top-10 documents that jump-start the CAL algorithm. Average is on 4 out of 7 topics.

1; hence, we show results averaged on 4 topics out of 7. We first show the Recall@10 (on the full reranking) in Table 5: notice how the zero-shot rankings effectively jump-start the CAL process from a higher recall; BioBERT proves to be the most effective algorithm to jump-start with. We show the WSS@{85, 95, 100}% and the MAP averaged across the topics in Table 6. Notice that the "ranking" here is actually the ordering of the documents collected at the end of the CAL process (CAL ordering). The results show how the higher initial recall translates to better performances on the average WSS scores with respect to the NP Logistic baseline, even though they are not consistently in line with the metrics taken on the zero-shot setup and the Recall@10: i.e., the top-10 documents coming from the DL Classifier or the pre-trained LR seem to be able to better jump-start the CAL process, despite BioBERT was the best model in terms of MAP (Table 4) and Recall@10 (Table 5). Furthermore, the DL Classifier was the worst of the three pre-trained models in both Tables 4 and 5, but the CAL process jumpstarted from its top-10 documents shows better WSS performances at the 95% thresholds than the other models. Regarding the MAP, the pre-trained models can effectively jump-start the CAL algorithm as seen for the WSS; notice that the top-10 documents from BioBERT manage to keep the advantage we saw in Table 4 for the MAP metric. From these results, overall, we conclude that the pre-training can actually improve on the baseline performances both in terms of MAP and WSS; however, as reported by [17] as well, knowledge transfer can fail in some cases.

6.2. RQ2: Can we keep training our DL models in the Active Learning process?

In our experiments so far, we have showed results on the zero-shot rankings from our models, or when using them to jump-start the CAL process. We did not, however, leverage the pre-trained DL models in the active learning

	CAL ordering			
	NP Logistic	DL Rank.	DL Class.	LR
WSS@85%	0.494 ± 0.241	0.504 ± 0.242	0.468 ± 0.234	**0.508** ± 0.254
WSS@95%	**0.475** ± 0.304	0.460 ± 0.321	**0.475** ± 0.204	0.460 ± 0.334
WSS@100%	0.372 ± 0.306	0.249 ± 0.207	0.297 ± 0.131	**0.378** ± 0.304
MAP	0.357 ± 0.134	0.351 ± 0.120	0.368 ± 0.138	**0.399** ± 0.133

Table 7: WSS@{85, 95, 100}% and MAP for the CAL orderings where we keep training the DL models inside the CAL process. Notice that the LR is not continuously trained and results are the same as reported in Table 6. Average is still on 4 topics out of 7.

process: can these models actually be trained in such a scenario? To understand this, we run another set of experiments with the same setup as before, but where we actually keep training our DL models during the active learning review process. Training a DL model in such a scenario is not a trivial task, since many hyperparameters have to be taken into account: epochs, cross-entropy class weights (to counteract class imbalance) and learning rate are just some of the hyperparameters we deal with. Regarding epochs, [15] fine-tune BERT in the AL process for 10 and 30 epochs (based on the dataset), albeit with no clear rationale behind the choice of the number of epochs; however, they also point out how crucial it is to have "just-right" tuning of the model.

Lacking a validation set, however, we run a first batch of experiments where we arbitrarily set these hyperparameters. Due to the high computational costs of fine-tuning BioBERT at every iteration, we decided against using it in this part of the experiments for RQ2; moreover, we argue that these very large language models are impractical to fine-tune in such a scenario, both due to their computational costs and to the disproportion between the high number of parameters to fine-tune and the size of training data. Regarding the DL Ranker and Classifier, we:

- train the models for 50 epochs at each CAL iteration;
- keep the class weights in the Cross-entropy loss at 0.2 and 0.8 for the negative and positive class respectively;
- use a learning rate of 0.001.

We show the results of such experiments in terms of WSS and MAP on the CAL ordering (Table 7). As we can clearly see from the table, continuously training these models during the CAL process has inconsistent effects on the metrics: with respect to the jump-started CAL results (Table 6), the DL Ranker only improves for the WSS@85% metric, showing slight to substantial decrease in performances for all other metrics. The DL Classifier is no different and exhibits a consistent loss of performances for all metrics. In summary, fine-tuning these models in an active learning process seems unadvisable: we think this might be due to (i) the small number of documents we usually have for fine-tuning (especially in the first CAL iterations), (ii) the training set size constantly changing (possibly too slowly), (iii) a number of parameters to update which is too large with respect to the training data, (iv) many hyperparameters which might need better adjustment in such a scenario.

We believe that a much better solution in this case might be to employ Adapter modules [23], freezing the rest of the network. This also allows us, in terms of computational costs, to fine-tune BioBERT.

Notice. Given the amount of hyperparameters involved, the absence of a validation set, and the small set of training topics, we have decided to show the impact of different hyperparameters directly on the testing topics, when using (and not using) adapter layers. These results should serve as a basis for further research on the matter and as a mean to better understand whether it is possible at all to properly train such big models (especially in BioBERT case) in a CAL setting, where the overall number of documents span from a few hundreds to a few thousands.

6.3. Hyperparameter search

As mentioned, we conduct a hyperparameter search study where we analyse the variation in Mean Average Precision due to the learning rate, the number of epochs and the percentage of documents assessed at every CAL iteration. We conduct this hyperparameter search directly on the testing topics: these experiments should be taken as an effort to understand why the DL models failed when continuously trained (see Section 6.2) and, possibly, where to look for a solution in future works; in other words, the aim of these experiments is not to compete with a baseline (which would not be fair, since we are testing hyperparameters directly on the test set), but rather to show the most promising directions to take in order to enable DL models to be actively trained. For this reason, we sometimes omit results when they are not particularly interesting (i.e., not exhibiting a pattern that we might exploit in the future) as to avoid cluttering the paper with too many figures.

That said, we evaluate the effect of these hyperparameters both when training the whole neural network and when using adapter layers. For the former case, we show results for the DL Ranker only, as it was the best DL zero-shot model (not considering BioBERT). For the latter case, we also show BioBERT results where we vary the learning rate. We test with different configurations:

- the learning rate values range from the default value used in training of 1×10^{-3}, to 1×10^{-5}. Being BioBERT a completely different model, we test here with the default learning rate[10] of 5×10^{-5} and the value suggested by the AdapterHub library[11] of 1×10^{-4}. Epochs are fixed at 60 for the Ranker and at 5 for BioBERT;

[10]This was one of the learning rates used in [16] and is the default in the HuggingFace library.

[11]`https://docs.adapterhub.ml/training.html`

- for the DL Ranker only, we also test the model by training for 10, 30, 60 and an adaptive number of epochs (see below) at every CAL iteration. Learning rate is fixed at 1×10^{-4};
- finally, we also train the DL Ranker annotating 5% and 20% of the documents at every CAL iteration. We indicate the percentage of documents we take at every iteration with Δ_d. BioBERT is fine-tuned with $\Delta_d = 5\%$ only.

By "adaptive number of epochs" we mean that the number of epochs change at every active learning iteration, as a function of the number of training documents we have collected so far. For these experiments, we have empirically defined this as:

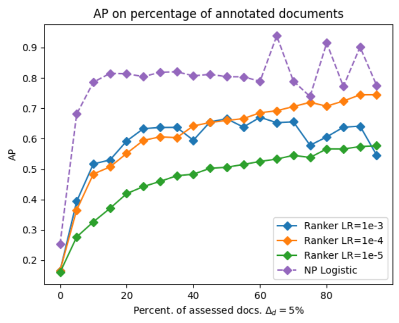
$$\text{epochs} = \min(|\mathcal{L}_i| \cdot 0.3, 500) \tag{4}$$

where with $|\mathcal{L}_i|$ we indicate the size of the available training documents at a given iteration i. That is, the number of epochs is equal to 30% of the training documents, with an upper bound set at 500 (the number of epochs used during the pre-training of the models).

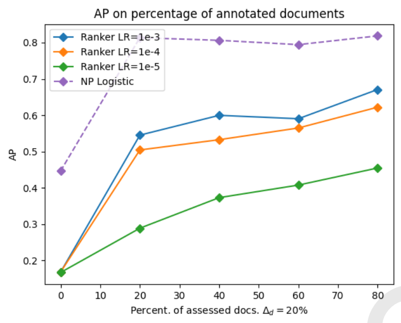
For the DL Ranker without adapter layers, we show the MAP for the different learning rate setups; we also show the NP Logistic as a baseline. The average is on all testing topics (as opposed to results in Section 6.1 and 6.2). More precisely, we continuously train and evaluate the models with the following procedure:

1. At iteration $i = 0$ (i.e., no document has been reviewed yet), all documents are ranked according to the pre-trained model zero-shot ranking;
2. We compute the AP of this ranking;
3. We review the top Δ_d documents and re-train the model;
4. We re-rank the whole pool of documents again and re-compute AP;
5. We repeat this process until all documents have been reviewed.

Clearly, the NP logistic is an exception, using the systematic review topic query as the initial seed document and following [9] procedure (as it did so far in our experiments, unless otherwise stated). In other words, at each iteration we take the full reranking (not the CAL ordering) and evaluate it. This is useful to understand whether the models under examination can indeed learn and improve on the previous iteration. Since we cannot exactly take 5% or 20% of the documents for all topics, we bin the results by number of annotated documents and plot the average of the bins. The results are plotted in Figure 2. When $\Delta_d = 5\%$, we notice how the default learning rate of 1×10^{-3} causes instability for the Mean Average Precision as the training set grows. The other two learning rates seem to be much more stable across CAL iterations, and a learning rate of 1×10^{-4} is capable of achieving MAP values close to the baseline's at later stages of the reviewing process. That said, the NP Logistic baseline is clearly the better algorithm, achieving and keeping a higher MAP across all iterations. Moreover, Figure 2b shows that reviewing 20% of the

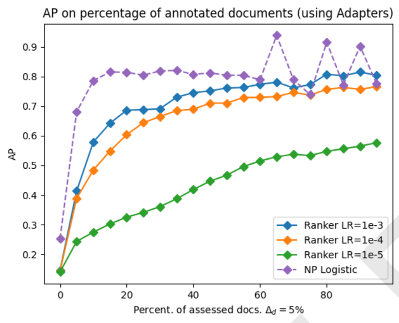


(a)

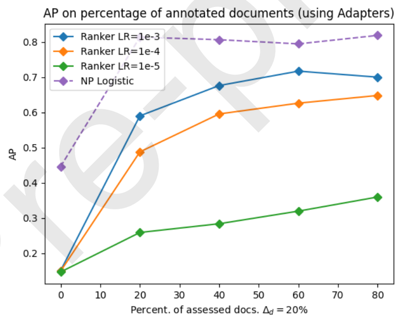


(b)

Figure 2: Variation of Mean Average Precision with different learning rates, annotating 5% (left) and 20% (right) of the documents at each iteration.



(a)



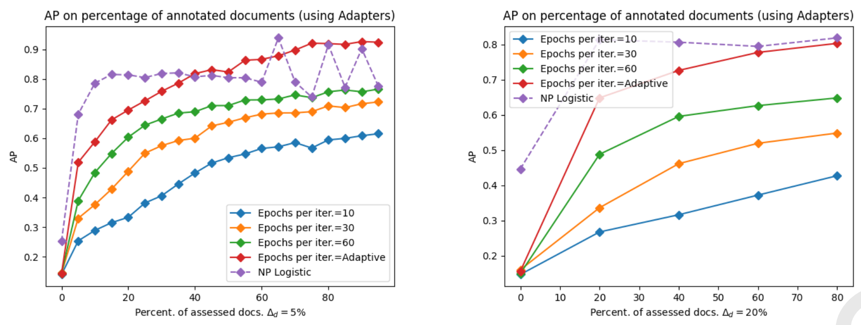
(b)

Figure 3: Variation of the Average Precision with different learning rates, annotating 5% (left) and 20% (right) of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.

documents at every iteration is suboptimal, leading to much lower values of MAP across all iterations.

Regarding adapter layers, we show the results in Figure 3 for the DL Ranker. The plots show how, when using adapters and $\Delta_d = 5\%$, a higher learning rate is able to achieve better performances. As a matter of fact, a learning rate of 1×10^{-3} obtains greater values of MAP than the baseline, at later stages of the CAL process. Overall, adapter layers seem to bring greater stability to the learning capability of the model (which is expected, having less parameters to learn). Finally, we notice once again how using $\Delta_d = 20\%$ brings to overall worse performances than with $\Delta_d = 5\%$.

We will now analyse the effect of the number of epochs on the performances of the DL Ranker. As mentioned before, we test with 10, 30, 60 and an empirically defined adaptive strategy (see Equation 4), that we call "Adaptive" in the plots. Since results for the DL Ranker without adapters were, similarly to the learning



(a) (b)

Figure 4: Variation of the Average Precision with different epochs, annotating 5% (left) and 20% (right) of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.

rate ones, not particularly interesting, we show the variation of MAP when using adapters only (Figure 4). As we have seen for the learning rate figures, using adapter layers can indeed bring to a substantial improvement on the average precision metric. As a matter of fact, the adaptive number of epochs can, at later stages, achieve a better MAP than the NP Logistic baseline; again, we notice that overall the gain in performance is much more consistent with the growth in training set size when using adapters[12]. In conclusion, we could say that, especially when using adapters, the number of epochs is a particularly sensitive hyperparameter (with respect to learning rate) which must be correctly adapted to the growing size of the training set; we believe future research on this topic might give new and interesting prospectives on the trainability of DL models in active learning scenarios.

Regarding the fine-tuning with adapters of the BioBERT algorithm, we only experimented with two different learning rates: (i) the default BioBERT learning rate in the HuggingFace library, i.e. 5e-5; (ii) the default learning rate in the AdapterHub library, i.e. 1e-4. The number of epochs, on the other hand, is fixed at 5 and $\Delta_d = 5\%$. This was done mainly for computational reasons, since, even with adapter layers, fine-tuning BioBERT can be a very expensive operation. Moreover, the results we were seeing from this initial set of experiments were not promising enough, and we decided against running further experimentation. As a matter of fact, looking at BioBERT results in Figure 5 we notice very poor performances, raising the question whether it is actually possible at all to train large language models when the dataset is relatively small and the update is done in a continual fashion: indeed, despite testing with two very different learning rates, the results seem to be just slightly affected, with 5×10^{-5} being the better

[12]We also point out that this setup can achieve WSS@95% close to the baseline, albeit not as consistently as the baseline can.

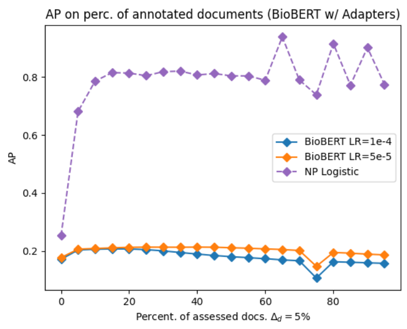


Figure 5: Variation of the Mean Average Precision with different learning rates for BioBERT, annotating 5% of the documents at each iteration. We only train Adapter layers and freeze the rest of the network.

of the two, even though not significantly. That said, further experimentation with different number of epochs might give more promising results.

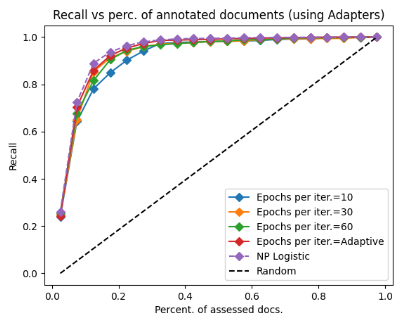
Finally, so far we have shown metrics on the full reranking, but we have not shown their CAL's ordering (which is what a reviewer would actually see). We plot how these orderings change as a function of the number of epochs (or learning rate, in BioBERT case) when $\Delta_d = 5\%$ and using adapter layers, to keep the number of plots to a minimum. We can see these orderings in Figure 6. Unsurprisingly, these plots show similar results to the previous ones, with the adaptive epochs obtaining the best results. That said, differences between the several epoch values tested are much smaller[13]. Furthermore, as we were seeing for the plots on BioBERT rankings, its CAL's ordering is also showing rather poor performances.

Finally, one critical aspect to consider when further studying the applicability of large language models to TAR is also the unavoidable increase in training times: at each active learning iteration we need to re-train the model, whose cost, when dealing with so many parameters, can be non-negligible. Indeed, even when only training adapters as we did here, training times can substantially increase: for completing the experiments (on all the testing topics), BioBERT with adapters took about 20 hours; the DL ranker with adapters needed less than 7 minutes and the LR just 20 seconds.

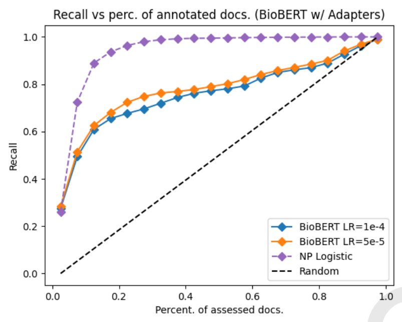
In conclusion, this hyperparameter search can help us give a first tentative picture of what works and what does not, as well as finding directions for future works:

1. deep learning models, be them very large models or tinier ones, cannot be simply updated in an active learning process. Despite starting from a more or less good zero-shot capability, their performances quickly deteriorate

[13]This is somehow expected, since, at every iteration, the updated model can only have an impact on the top-k documents reviewed in the next batch, and not on the previous ones.



(a)



(b)

Figure 6: Variation on recall vs percentage of assessed documents due to different learning rates. Models have been trained with $\Delta_d = 5\%$. We show the CAL ordering of the different continuously trained models.

when trained in these scenarios;

2. adapter layers can be a good solution for fine-tuning, except when the underlying frozen model is excessively large (especially compared to the size of the dataset). Indeed, models such as BioBERT are rather good at transfer learning, but cannot seemingly be fine-tuned in the CAL process;
3. some of the hyperparameters can be of key importance to the success of the models fine-tuning. Understanding how to adapt hyperparameters such as the learning rate and the number of epochs to the increasing training set size, as well as being able to assess how many documents are reviewed at each iteration, can make the difference between a decent model that can compete with current state of the art and a rather poor one.

Regarding our bullet point 2, it would be interesting to explore, in future works, which type of adapter layers (e.g., [23, 24, 25]) can bring about the most promising increase in performances and if the peculiar active learning scenario might require further adaptations or modifications of these techniques to fully leverage the zero-shot knowledge that we were seeing in Tables 4 and 5.

Finally, for bullet point 3, the correct choice of hyperparameters can be truly problematic since we lack a validation set. It could indeed be possible to extract a validation set before starting the review process, but this seems to make sense only when the dataset is large enough, and should be compared to a baseline which is also taking into account the presence of such a validation set. That said, it would be interesting to explore whether a more or less empirical solution can be found, which could allow selecting and/or adapting the hyperparameters without necessarily looking at a validation set; to this end, adapting the number of epochs to the size of the current training set seems to be particularly effective and should be further explored.

7. Conclusion

In this work, we explored whether using previous Systematic Reviews (SR) to pre-train machine learning models can actually bring better performances for a new SR topic, compared to doing no pre-training at all. Specifically, we also investigated whether deep learning models such as BioBERT or other transformer-like architectures can be effective, and to which extent. We conducted experiments on the CLEF TAR 2019 Task 2 dataset, and the results clearly show that pre-trained models can obtain good zero-shot rankings on both the Mean Average Precision and Work Saved over Sampling metrics (Section 6.1). When used with the CAL algorithm, we also see that jump-starting the active learning process from these zero-shot rankings can actually bring to a higher recall earlier in the assessment process (Section 6.2). Finally, we also noticed how continuously training our deep learning models brings to inconsistent performances (usually, with a detrimental effects on the evaluation metrics): we then conducted an extensive analysis on a hyperparameter search (Section 6.3). The aim of this latter experimentation was to understand how and what we would need to change (or further research) to effectively train deep learning models in an active learning process. Our results show that future works should focus on finding and (at least empirically) define a set of rules to adapt hyperparameters (e.g. epochs and learning rate) to the growing training set size; more specifically, we believe that using smaller models and implementing adapter modules can bring substantial improvements over the standard non-pre trained logistic regression, if paired with a proper adaptation of the number of epochs to the training set size.

References

[1] I. Shemilt, N. Khan, S. Park, J. Thomas, Use of cost-effectiveness analysis to compare the efficiency of study identification methods in systematic reviews, Systematic reviews 5 (1) (2016) 1–13.

[2] M. Michelson, K. Reuter, The significant cost of systematic reviews and meta-analyses: a call for greater involvement of machine learning to assess the promise of clinical trials, Contemporary clinical trials communications 16 (2019) 100443.

[3] G. Tsafnat, P. Glasziou, M. K. Choong, A. Dunn, F. Galgani, E. Coiera, Systematic review automation technologies, Systematic reviews 3 (1) (2014) 1–15.

[4] H. Scells, G. Zuccon, Generating better queries for systematic reviews, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 2018, pp. 475–484.

[5] H. Scells, G. Zuccon, B. Koopman, J. Clark, Automatic Boolean Query Formulation for Systematic Review Literature Search, in: Proceedings of

The Web Conference 2020, ACM, Taipei Taiwan, 2020, pp. 1071–1081. doi:10.1145/3366423.3380185.
URL https://dl.acm.org/doi/10.1145/3366423.3380185

[6] G. V. Cormack, M. R. Grossman, Technology-Assisted Review in Empirical Medicine: Waterloo Participation in CLEF eHealth 2017, in: CLEF (working notes), 2017, p. 11.

[7] G. E. Lee, A. Sun, Mirror Matching: Document Matching Approach in Seed-driven Document Ranking for Medical Systematic Reviews, arXiv:2112.14318 [cs]ArXiv: 2112.14318 (Dec. 2021).
URL http://arxiv.org/abs/2112.14318

[8] G. E. Lee, A. 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, ACM, Ann Arbor MI USA, 2018, pp. 455–464. doi:10.1145/3209978.3209994.
URL https://dl.acm.org/doi/10.1145/3209978.3209994

[9] G. V. Cormack, M. R. Grossman, Autonomy and reliability of continuous active learning for technology-assisted review, arXiv preprint arXiv:1504.06868 (2015).

[10] E. Yang, D. D. Lewis, O. Frieder, Heuristic stopping rules for technology-assisted review, in: Proceedings of the 21st ACM Symposium on Document Engineering, 2021, pp. 1–10.

[11] D. Li, E. Kanoulas, When to stop reviewing in technology-assisted reviews: Sampling from an adaptive distribution to estimate residual relevant documents, ACM Transactions on Information Systems (TOIS) 38 (4) (2020) 1–36.

[12] A. M. Cohen, K. Ambert, M. McDonagh, Cross-topic learning for work prioritization in systematic review creation and update, Journal of the American Medical Informatics Association 16 (5) (2009) 690–704. doi:https://doi.org/10.1197/jamia.M3162.
URL https://www.sciencedirect.com/science/article/pii/S1067502709001224

[13] A. Lagopoulos, G. Tsoumakas, From protocol to screening: A hybrid learning approach for technology-assisted systematic literature reviews, arXiv preprint arXiv:2011.09752 (2020).

[14] J. Pickens, On the effectiveness of portable models versus human expertise under continuous active learning, in: 2nd International Workshop on AI and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA), 2021.

[15] E. Yang, S. MacAvaney, D. D. Lewis, O. Frieder, Goldilocks: Just-right tuning of bert for technology-assisted review, arXiv preprint arXiv:2105.01044 (2021).

[16] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[17] H. Zhao, S. Ye, J. Yang, An empirical study on transfer learning for privilege review, in: 2021 IEEE International Conference on Big Data (Big Data), IEEE, 2021, pp. 2729–2733.

[18] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, J. Kang, Biobert: a pre-trained biomedical language representation model for biomedical text mining, Bioinformatics 36 (4) (2020) 1234–1240.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, arXiv preprint arXiv:1706.03762 (2017).

[20] P. Pobrotyn, T. Bartczak, M. Synowiec, R. Białobrzeski, J. Bojar, Context-Aware Learning to Rank with Self-Attention, arXiv:2005.10084 [cs]ArXiv: 2005.10084 (Jul. 2020).
URL `http://arxiv.org/abs/2005.10084`

[21] P. Pobrotyn, R. Białobrzeski, NeuralNDCG: Direct Optimisation of a Ranking Metric via Differentiable Relaxation of Sorting, arXiv:2102.07831 [cs]ArXiv: 2102.07831 (Feb. 2021).
URL `http://arxiv.org/abs/2102.07831`

[22] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[23] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, S. Gelly, Parameter-efficient transfer learning for nlp, in: International Conference on Machine Learning, PMLR, 2019, pp. 2790–2799.

[24] J. Pfeiffer, I. Vulić, I. Gurevych, S. Ruder, Mad-x: An adapter-based framework for multi-task cross-lingual transfer, arXiv preprint arXiv:2005.00052 (2020).

[25] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, G. Neubig, Towards a unified view of parameter-efficient transfer learning, arXiv preprint arXiv:2110.04366 (2021).

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Alessio Molinari: Formal analysis; Funding acquisition; Investigation; Methodology; Resources; Software; Supervision; Validation; Visualization; Roles/Writing - original draft; Writing - review & editing

Evangelos Kanoulas: Formal analysis; Funding acquisition; Investigation; Methodology; Project administration; Resources; Supervision; Validation; Visualization; Roles/Writing - original draft; Writing - review & editing