Capreolus: A Toolkit for End-to-End Neural Ad Hoc Retrieval

Andrew Yates,[1] Siddhant Arora,[2] Xinyu Zhang,[3] Wei Yang,[3] Kevin Martin Jose,[1,4] Jimmy Lin[3]
[1] Max Planck Institute for Informatics, Saarland Informatics Campus
[2] Indian Institute of Technology, Delhi
[3] David R. Cheriton School of Computer Science, University of Waterloo
[4] Saarland University, Saarland Informatics Campus

ABSTRACT

We present *Capreolus*, a toolkit designed to facilitate end-to-end *ad hoc* retrieval experiments with neural networks by providing implementations of prominent neural ranking models within a common framework. Our toolkit adopts a standard reranking architecture via tight integration with the Anserini toolkit for candidate document generation using standard bag-of-words approaches. Using Capreolus, we are able to reproduce Yang et al.'s recent SIGIR 2019 finding that, in a reranking scenario on the test collection from the TREC 2004 Robust Track, many neural retrieval models do not significantly outperform a strong query expansion baseline. Furthermore, we find that this holds true for five additional models implemented in Capreolus. We describe the architecture and design of our toolkit, which includes a Web interface to facilitate comparisons between rankings returned by different models.

ACM Reference Format:
Andrew Yates, Siddhant Arora, Xinyu Zhang, Wei Yang, Kevin Martin Jose, and Jimmy Lin. 2020. Capreolus: A Toolkit for End-to-End Neural Ad Hoc Retrieval. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3336191.3371868

1 INTRODUCTION

Neural approaches to *ad hoc* document retrieval have become popular alternatives to learning-to-rank techniques based on massive feature engineering, but the architecture of search systems still remains dominated by a multi-stage reranking model. In this pipeline approach, an initial list of candidate documents (typically retrieved with a bag-of-words term-matching technique using an inverted index) is passed to one or more reranker stages that are based on neural models.

In this context, information retrieval researchers have a long history of developing shared, open-source IR toolkits, such as Indri [14], Terrier [10], and Anserini [17]. In a multi-stage pipeline architecture, these systems can provide the initial candidate list. This open-source tradition has carried over to neural ranking models—it is quite commonplace to find research papers paired with open-source implementations of the proposed models. This has led to a proliferation of individual model implementations that differ in both neural training decisions (e.g., sampling strategies) and in earlier stages of the pipeline, making cross-model comparisons of ranking effectiveness difficult.

Recently, the MatchZoo toolkit [5] addressed the first part of this issue by providing implementations of general text matching models, including a number of *ad hoc* retrieval models. In order to remain general, MatchZoo (as well as other toolkits) focus on the general text matching task and consider much of an end-to-end experimental IR pipeline to be out of scope. As modern experimental research pipelines have increased in complexity, the amount of code needed to conduct an experiment has also increased. Such code contains lots of repetitive "boilerplate" and is error prone when replicated across many different individual experiments.

For example, consider the common scenario of using a cross-validated neural model to rerank documents from a baseline bag-of-words term-matching scoring function like BM25. This scenario requires tuning of both the baseline scoring function's parameters (which can differ depending on the training fold) and a large number of neural hyperparameters, including both hyperparameters specific to the model (e.g., CNN filter sizes) and general neural hyperparameters whose optimal values can vary with the model (e.g., loss function, which pretrained embeddings to use, learning rate, batch size, iterations to train for, micro vs. macro sampling of queries, etc). Such choices vary with each individual model implementation, making it necessary for researchers to either re-implement the models in a common framework or to identify and resolve differences before the models can be fairly compared.

In this work, we present *Capreolus*,[1] a toolkit designed to facilitate end-to-end IR experiments, particularly large-scale comparative studies, by providing PyTorch implementations of ten prominent neural retrieval models in a common framework. Capreolus tightly integrates with the Anserini IR toolkit [17] for initial candidate list generation. This approach enables fair comparisons between models by allowing the entire pipeline to be controlled—from indexing raw documents to producing rankings and cross-validated metrics—thus filling the gaps in existing neural pipelines.

Contributions. Our work makes the following contributions:

- the open-source Capreolus toolkit[2] for neural *ad hoc* retrieval, which includes implementations of ten prominent neural retrieval models in a common framework with Anserini integration for candidate list generation;
- a reproduction of the *Neural Hype* paper by Yang et al. [18], supporting the authors' finding that many neural retrieval models

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371868

[1] https://mpi-inf.mpg.de/departments/databases-and-information-systems/research/neural-ir/capreolus
[2] https://capreolus.ai

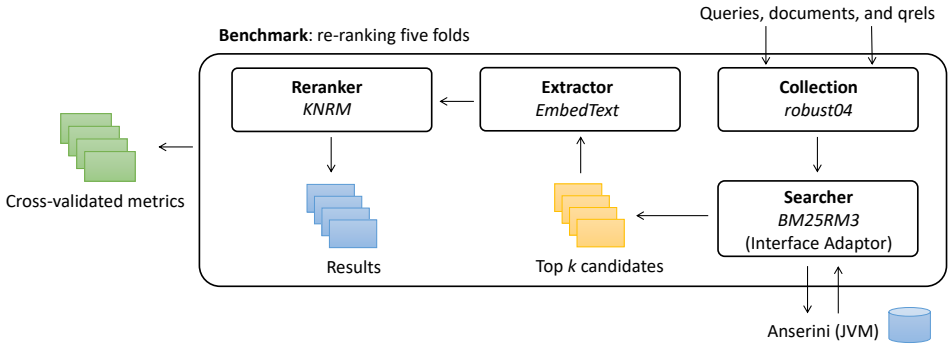

Figure 1: The architecture of Capreolus. A pipeline for KNRM is shown, with candidate documents from the Robust04 collection provided by Anserini, which are then reranked with KNRM. These steps are repeated over n folds by the *benchmark*. Module names are in bold (e.g., Reranker, Extractor) with concrete class names below them in italics (e.g., KNRM, EmbedText).

do not significantly outperform a strong query expansion baseline on the test collection from the TREC 2004 Robust Track (Robust04); and,

- a demonstration consisting of a Web interface, allowing users to interactively query a collection and compare results between retrieval models under different configurations.

2 CAPREOLUS FRAMEWORK

In this section we describe the architecture of Capreolus, which is organized around five modules: a collection, a searcher, an extractor, a reranker, and a benchmark. These modules are responsible for providing raw queries and documents (*collection*), tuning a first-stage ranking method and using it to provide a list of candidate documents (*searcher*), transforming the query and candidate documents into inputs suitable for a reranking model (*extractor*), reranking the candidate documents (*reranker*), and managing the data splits used in the previous steps to calculate cross-validated metrics (*benchmark*). In the following sections we describe Capreolus' modules, training process, and Web interface in more detail.

2.1 Modules and Configuration

Capreolus consists of five main modules, which dictate the steps required to train and evaluate a neural reranking model. Figure 1 illustrates the data flow between modules. While many of the modules are of general utility, they are primarily intended to be used through a Python script that takes configuration options and runs all modules accordingly (this includes loading the desired collection, retrieving a first-stage ranking, training a neural model, and producing results on test queries).

Each module is associated with its own configuration options that may be provided on the command line or in a YAML file. Non-neural modules are deterministic given their inputs and configuration options, allowing Capreolus to cache and re-use intermediate outputs (e.g., tuned BM25 results). Below, we describe each module in greater detail.

Collection. A *collection* describes the location of raw documents, queries, and relevance judgments (*qrels*) on disk, as well as the information needed to process them (e.g., the documents' format). This module exposes only one configuration option: the name of the collection to use, such as *robust04* or *gov2*. New collections can be added to Capreolus by creating a YAML file containing the paths to the associated files and their formats.

Searcher. The *searcher* module is responsible for identifying the top k candidate documents for each query so that they may be reranked. Static and cross-validated searcher types are supported. A static searcher simply queries Anserini with fixed parameters to return the candidate documents (e.g., BM25 with $k_1 = 0.9$ and $b = 0.4$), whereas a cross-validated searcher uses the validation set to tune a ranking method's parameters before using these parameters to return the candidate documents. Candidates obtained with either searcher type are cached. Capreolus provides several searcher modules that the user may choose from, including static and tuned variants of BM25 and RM3. Users may additionally configure the associated preprocessing, such as whether stemming and stopword removal are applied. While all existing searcher modules are powered by Anserini, the backend is designed to be interchangeable and searcher modules for other search engines could be developed without substantial effort.

Extractor. The *extractor* module is responsible for transforming candidate documents (and the associated queries) into representations suitable for the selected neural model. Each reranking model specifies the extractor that it requires, because extractors may be closely tied to the neural model (e.g., input to DeepTileBars requires segment tokenization). Reranking models that consume similarity matrices are coupled with the general `EmbedText` extractor, which tokenizes the input text and then provides two inputs: a query-document similarity matrix constructed using embeddings and the IDF of each query term. Configuration options are specific to the extractor selected, but include choices like the type of embeddings to use and whether stopwords should be removed.

Reranker. The *reranker* module is responsible for reranking the candidate documents. Capreolus provides implementations of ten neural reranking models for *ad hoc* retrieval. The rerankers available include *representation-focused models* (DSSM [6], CDSSM [13]), *interaction-focused models* (DRMM [4], KNRM [16], PACRR [7], ConvKNRM [2], POSITDRMM [12]), *hybrid models* (DUET [11]), and *hierarchical models* (HiNT [3] and DeepTileBars [15]). Each reranker is associated with model-specific configuration options

(e.g., the number of histogram bins to use with DRMM or the size of ConvKNRM's kernels) as well as configuration options specific to the neural training process, such as the sampling strategy, learning rate, and random seed. A model's optimal hyperparameters may vary with both the collection and the implementation, making it essential that they can be tuned with minimal effort in a new experimental setup.

Benchmark. A *benchmark* defines training, validation, and testing folds, which are used to evaluate the pipeline's ranking effectiveness. This generally requires running the modules on each data split and aggregating the results. Users may configure the data splits used (e.g., to match folds from prior work) and metrics reported.

2.2 Training and Evaluation

Given a pipeline configured with the above modules, Capreolus performs any needed preprocessing (e.g., creating an Anserini index and determining the searcher's optimal parameters for the given fold) before beginning a training loop. Inside the training loop, a neural optimizer is used with a pairwise loss like hinge loss to train the model. The model's effectiveness is evaluated on the validation set and the best performing weights are saved for later use, along with a corresponding TREC-format run file. Additionally, training loss and validation set metrics are plotted once training completes. To evaluate the trained model, the best weights on the validation set are used to calculate the desired metrics on the test set. The benchmark may aggregate these metrics over multiple folds, depending on the experimental setup.

2.3 Web Interface

Capreolus provides a Web interface for interactively comparing the rankings produced by a pair of retrieval models (or a pair of configurations for the same model). The user can choose model configurations to compare, query a collection with them, and examine the reranking results returned by the two models. Alternatively, to compare a first-stage ranking with the reranked results, the user can choose a non-neural baseline to serve as one of the models being compared. The search engine results page allows the user to view the documents returned and the ranks given to the documents by each model. When relevance judgments are available, the results page also indicates each document's relevance.

3 EXPERIMENTS

As a demonstration of Capreolus, we conduct experiments reproducing a recent finding: Yang et al. [18] showed that many neural IR models cannot beat a well-tuned query expansion baseline in the "low data" scenario on Robust04. We replicate the experimental setup used in the paper, confirming their findings and presenting results on five additional models[3] provided by Capreolus.

3.1 Experimental Setup

Following Yang et al. [18], we conduct experiments on the TREC Robust04 collection, and use BM25 with RM3 (tuned) to retrieve 1000 candidate documents to rerank for each query. As in the original work, we use 5-fold cross-validation with three folds for training, one for validation, and the remaining fold for evaluation.[4] We report Mean Average Precision (MAP), P@20, and nDCG@20.

For the neural rerankers, which require input of a fixed size, we remove stopwords, set the maximum query length to 4 tokens, and set the maximum document length to 800 tokens. For the rerankers that use term embeddings, we employ GloVe 6B embeddings with 300 dimensions and freeze the embedding layer. Unless otherwise mentioned, each reranking model is trained using the Adam optimizer [8] with a learning rate of 10^{-3} for 50 iterations of 4,096 training instances using a batch size of 32. We use hinge loss with a margin of 1.0. The model-specific hyperparameters are as follows:

- **CDSSM.** We reduce the model's size by using only 1 filter in the CNN. Both the kernel size and feed forward layer's size were set to 30. To improve effectiveness, we replace the character-trigram inputs with term embeddings and set the window size to 4.
- **ConvKNRM.** We use 300 filters in the convolution layers and the same kernel functions as in the original paper, but treat μ_k and σ_k as trainable parameters.
- **DeepTileBars.** For the TextTiling algorithm, the passage length and window size were set to 30 and 6, respectively. For the model, we set $l = 7$, the number of CNN filters to 3, the LSTM size to 3, and used MLPs of size 32 and 16.
- **DRMM.** We use LogCount-based Histograms (LCH) and weight query terms with their IDF. The numbers of histogram bins and hidden nodes were set to 29 and 5, respectively.
- **DSSM.** We use one CNN layer with 56 filters to reduce overfitting. As in the original work, input is provided as character-trigrams.
- **DUET.** We used 10 filters with both the local and distributed model, with hidden dimensions set to 30 and 699, respectively.
- **HiNT.** We set the global decision layer's LSTM size to 6, $k = 10$, the learning rate to 0.005, and the batch size to 128.
- **KNRM.** We use the same kernel functions as the original paper, but treat μ_k and σ_k as trainable parameters.
- **PACRR.** We use 32 CNN filters and set $k = 2$ with k-max pooling.
- **POSITDRMM.** We implemented the multi-view POSITDRMM variant. This includes the features described in the original paper, except that we use BM25+RM3 in place of the BM25 score.

Following the original paper, we report interpolated results in order to examine the additivity of each neural model when combined with a non-neural ranking method (i.e., BM25+RM3). To do so, we use the interpolation formula:

$$\text{score} = \alpha \cdot \text{score}_{\text{Neural}} + (1 - \alpha) \cdot \text{score}_{\text{RM3}}$$

where the parameter α is tuned on the validation set. We additionally report results without interpolation for comparison.

3.2 Results

The results are shown in Table 1. The left half of the table presents results interpolated with the relevance score produced by BM25+RM3, which corresponds to the setup in Yang et al.'s original work, while the right half presents results without interpolation. We put these results in context by displaying additional results from the BERT-based Birch [20] (3S variant) and CEDR (KNRM variant) [9] neural

[3] Due to their much higher computational requirements, we leave results using BERT-based rerankers for future work.

[4] The folds used by Yang et al. are available at https://github.com/castorini/anserini/blob/master/src/main/resources/fine_tuning/robust04-paper2-folds.json.

Model	MAP	P@20	nDCG	MAP	P@20	nDCG
BM25+RM3	0.303	0.397	0.451			
Birch	0.369	0.467	0.533			
CEDR	-	0.467	0.538			
	Interpolation			No Interpolation		
POSITDRMM	0.301	0.396	0.449	0.299	0.396	0.447
PACRR	0.303	0.402	0.457	0.259*	0.358*	0.409*
DRMM	0.306	0.397	0.454	0.256*	0.340*	0.399*
KNRM	0.307	0.399	0.456	0.248*	0.331*	0.388*
ConvKNRM	0.297	0.396	0.455	0.184*	0.266*	0.306*
DeepTileBars	0.304	0.391	0.450	0.182*	0.277*	0.313*
HiNT	0.302	0.402	0.459	0.176*	0.255*	0.305*
DUET	0.303	0.395	0.451	0.132*	0.194*	0.226*
DSSM	0.301	0.394	0.451	0.067*	0.106*	0.111*
CDSSM	0.303	0.397	0.449	0.081*	0.121*	0.135*

Table 1: Results on the Robust04 dataset. The symbol * indicates significant difference ($p < 0.05$) based on a paired t-test between the reranker and baseline.

reranking models. To the best of our knowledge, these are currently the state-of-the-art neural models on Robust04.

We first evaluate our reproduction attempt using the interpolated results on the models considered by Yang et al. Of the five models considered in the original work (i.e., DRMM, KNRM, DUET, DSSM, and CDSSM), MAP and nDCG@20 for CDSSM, DSSM, and DUET are always within 1% of the original values (Yang et al. did not report P@20.) KNRM is 2.7% higher than the original work in terms of nDCG@20, while DRMM is 2.9% lower than the original work in terms of MAP and 3.8% in terms of nDCG@20. Similarly, our BM25+RM3 metrics exactly match those in the original work, though this is unsurprising given that Anserini's implementation was used in both settings. These results provide evidence that Yang et al.'s findings are robust, because our Capreolus implementations (in PyTorch) are completely independent from those from MatchZoo (in Tensorflow) used in the original paper. Thus, our experiments confirm that, leaving BERT aside, most neural ranking models do not significantly improve upon a tuned query expansion baseline when given only limited amounts of training data.

Next, we consider interpolated results for the five new models: POSITDRMM, PACRR, ConvKNRM, DeepTileBars, and HiNT. All five models perform similarly, with nDCGs from 0.449 to 0.459. Thus, the addition of these models does not change our finding.

Finally, we consider the results without interpolation. POSITDRMM, which uses the BM25+RM3 score as a feature, is the only reranking method that does not perform significantly worse than the baseline. As discussed in prior work [4], the interaction-focused models consistently outperform the representation-focused models (DSSM and CDSSM). DUET, a hybrid model with both representation-focused and interaction-focused components, performs substantially better than the purely representation-focused models.

4 CONCLUSION

While we have confirmed Yang et al.'s observation about the effectiveness of many neural ranking models, we have not answered the question of *why*. The most obvious explanation is that Robust04 is too small, with only 250 queries (though it contains over 300,000 judged query–documents pairs). Many of the models implemented in Capreolus have a large number of learnable parameters (e.g., POSITDRMM with 3.6 million); it is perhaps no coincidence that DRMM and KNRM, with only 149 parameters and 34 parameters, respectively, perform slightly better with interpolation. However, recent experiments with BERT-based rerankers [1, 9, 19] show impressive improvements on Robust04, even with limited data. These models, however, benefit from massive pretraining (albeit *without* relevance judgements). Untangling these various effects to gain a better understanding of "pre-BERT" and "post-BERT" neural models represents future work that we are currently pursuing.

ACKNOWLEDGMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada, and enabled by computational resources provided by Compute Ontario and Compute Canada. We thank Sean Macavaney for code contributions and gratefully acknowledge the support of NVIDIA Corporation with the donation of a Titan X Pascal GPU used in this research.

REFERENCES

[1] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *SIGIR*.
[2] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2019. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *AAAI*.
[3] Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. 2018. Modeling Diverse Relevance Patterns in Ad-hoc Retrieval. In *SIGIR*.
[4] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM*.
[5] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. 2019. MatchZoo: A Learning, Practicing, and Developing System for Neural Text Matching. In *SIGIR*.
[6] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *CIKM*.
[7] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. 2017. PACRR: A Position-Aware Neural IR Model for Relevance Matching. In *EMNLP*.
[8] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).
[9] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *SIGIR*.
[10] Craig Macdonald, Richard McCreadie, Rodrygo L.T. Santos, and Iadh Ounis. 2012. From Puppy to Maturity: Experiences in Developing Terrier. In *OSIR Workshop at SIGIR*.
[11] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *WWW*.
[12] Ion Androutsopoulos Ryan McDonald, Georgios-Ioannis Brokos. 2018. Deep Relevance Ranking Using Enhanced Document-Query Interactions. In *EMNLP*.
[13] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *CIKM*.
[14] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. 2005. Indri: A Language Model-Based Search Engine for Complex Queries. In *Proceedings of the International Conference on Intelligent Analysis*. 2–6.
[15] Zhiwen Tang and Grace Hui Yang. 2019. DeepTileBars: Visualizing Term Distribution for Neural Information Retrieval. In *AAAI*.
[16] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *SIGIR*.
[17] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *J. Data and Information Quality* 10, 4 (2018), Article 15.
[18] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. 2019. Critically Examining the Neural Hype: Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models. In *SIGIR*.
[19] Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Simple Applications of BERT for Ad Hoc Document Retrieval. *arXiv:1903.10972* (2019).
[20] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-Domain Modeling of Sentence-Level Evidence for Document Retrieval. In *EMNLP*.