Let's measure run time!

Extending the IR replicability infrastructure to include performance aspects

Sebastian Hofstätter
TU Wien
s.hofstaetter@tuwien.ac.at

Allan Hanbury
TU Wien
hanbury@ifs.tuwien.ac.at

ABSTRACT

Establishing a docker-based replicability infrastructure offers the community a great opportunity: measuring the run time of information retrieval systems. The time required to present query results to a user is paramount to the users satisfaction. Recent advances in neural IR re-ranking models put the issue of query latency at the forefront. They bring a complex trade-off between performance and effectiveness based on a myriad of factors: the choice of encoding model, network architecture, hardware acceleration and many others. The best performing models (currently using the BERT transformer model) run orders of magnitude more slowly than simpler architectures. We aim to broaden the focus of the neural IR community to include performance considerations – to sustain the practical applicability of our innovations. In this position paper we supply our argument with a case study exploring the performance of different neural re-ranking models. Finally, we propose to extend the OSIRRC docker-based replicability infrastructure with two performance focused benchmark scenarios.

1 INTRODUCTION

The replicability and subsequent fair comparison of results in Information Retrieval (IR) is a fundamentally important goal. Currently, the main focus of the community is on the effectiveness results of IR models. We argue that in the future the same infrastructure supporting effectiveness replicability should be used to measure *performance*. We use the term *performance* in this paper in the sense of speed and run time – for the quality of retrieval results we use *effectiveness*. In many cases, the time required to present query results to a user is paramount to the users satisfaction, although in some tasks users might be willing to wait longer for better results [20]. Thus a discussion in the community about existing trade-offs between performance and effectiveness is needed.

This is not a new insight and we don't claim to re-invent the wheel with this position paper, rather we want to draw attention to this issue as it becomes more prevalent in recent advances in neural network methods for IR. Neural IR ranking models are re-rankers using the content text of a given query and a document to assign a relevance score. Here, the choice of architecture and encoding model offer large effectiveness gains, while at the same time potentially impacting the speed of training and inference by orders of magnitude.

The recently released MS MARCO v2 re-ranking dataset [1] is the first public test collection large enough to easily reproduce neural IR models.[1] The public nature of the dataset also makes it the prime contestant for replicability efforts for neural IR models.

Nogueira et al. [13, 14] first showed the substantial effectiveness gains for the MS MARCO passage re-ranking using *BERT* [4], a large pre-trained transformer based model. However, they note the stark trade-off with respect to performance. MacAvaney et al. [11] show that by combining *BERT*'s classification label with the output of various neural models exhibits good results for low-training-resource collections. They also show that this comes at a substantial performance cost – *BERT* taking two orders of magnitude longer than a simple word embedding.

On one hand, the retrieval results achieved with *BERT*'s contextualized encoding are truly impressive, on the other hand, the community should not lose focus of the practicality of their solutions – requiring fast performance for search. Complementing our argument we present a case study about the performance of different neural IR models and embedding models (Section 2). We show that using a FastText [2] encoding provides a small trade-off between effectiveness and performance, whereas *BERT* shows a big trade-off in both directions. *BERT* is more than 100 times slower than non-contextualized ranking models.

The medical computer vision community has already recognized the need for a focus on run time considerations. The medical image analysis benchmark VISCERAL [10] included run time measurements of participant solutions on the same hardware. Additionally, computer vision tasks, such as object detection and tracking, often require realtime results [8]. Here, iterations over neural network architectures have been focusing on performance [18, 19]. The object detection architectures commonly start with a pre-trained feature extraction model. As Huang et al. [8] show, this feature extraction stage can easily be swapped to accommodate different performance-effectiveness needs. We postulate that for neural IR models the time has come to do the same. Neural IR models depend on an encoding layer and recent works [6, 11, 13] show that the neural IR community has at least 4 different encoding architectures to choose from (basic word embedding, FastText, ELMo, BERT).

The public comparison of results on leaderboards and evaluation campaigns sparks interest and friendly competition among researchers. However, they naturally incentivise a focus on the effectiveness metrics used and other important aspects of IR systems – for example the latency of a response – are left aside. The introduction of docker-based submissions of complete retrieval systems makes the comparison of run time metrics feasible: All system can be compared under the same hardware conditions by a third party.

---

[1]Easy in this context means: MS MARCO has enough training samples to successfully train the neural IR models without the need of a bag of tricks & details applied to the pre-processing & training regime – which are often not published in the accompanying papers

Concretely, we propose to extend the docker-based replicability infrastructure for two additional use cases (Section 3):

(1) **Dynamic full system benchmark**
    We measure the query latency and throughput over a longer realistic period of a full search engine (possibly including a neural re-ranking component). We envision a scripted "interactive" mode, where the search engine returns results for a single query at a time, giving the benchmark a lot of fidelity in reporting performance statistics.

(2) **Static re-ranking benchmark**
    We measure the (neural) re-ranking components in isolation, providing them with the re-ranking candidate list. This allows for direct comparability of models as all external factors are fixed. This static scenario is very close to the way neural IR re-ranking models are evaluated today, with added timing metrics.

A standardized performance evaluation helps the research community and software engineers building on the research to better understand the trade-offs of different models and the performance requirements that each of them have. It is our understanding that the replicability efforts of our community are not only for *good science*, they are also geared towards the usability of our innovations in systems that people use. We argue that the performance is a major contributor to this goal and therefore worthwhile to study as part of a broader replicability and reproducibility push.

2 NEURAL IR MODEL PERFORMANCE

In the following case study we take a closer look at the training and inference time as well as GPU memory requirements for different neural IR models. Additionally, we compare the time required to re-rank a query with the model's effectiveness.

2.1 Neural IR Models

We conduct our experiments on five neural IR models using a basic Glove [17] word embedding and FastText [2]. Additionally, we evaluate a *BERT* [4] based ranking model. We use the MS MARCO [1] passage ranking collection to train and evaluate the models. All models are trained end-to-end and the word representations are fine-tuned. Now, we give a brief overview of the models used with a focus on performance sensitive components:

KNRM [22] applies a differentiable soft-histogram (Gaussian kernel functions) on top of the similarity matching matrix of query and document tokens – summing the interactions by their similarity. The model then learns to weight the different soft-histogram bins.

CONV-KNRM [3] extends *KNRM* by adding a Convolutional Neural Network (CNN) layer on top of the word embeddings, enabling word-level n-gram representation learning. *CONV-KNRM* cross-matches n-grams and scores n^2 similarity matrices in total.

MatchPyramid [15] is a ranking model inspired by deep neural image processing architectures. The model first computes the similarity matching matrix, which is then applied to several stacked CNN layers with dynamic max-pooling to ensure a fixed size output.

PACRR [9] applies different sized CNN layers on the match matrix followed by a max pooling of the strongest signals. In contrast to MatchPyramid, the CNNs are only single layered, focusing on different n-gram sizes and single word-to-word interactions are modeled without a CNN.

DUET [12] is a hybrid model applying CNNs to local interactions and single vector representation matching of the query and document. The two paths are combined at the end of the model to form the relevance score. *Note: We employed v2 of the model. We changed the local interaction input to a cosine match matrix – in line with the other models – in contrast to the exact matching in the published DUET model. We were not able to reproduce the original exact match results, however the cosine match matrix shows significantly better results than in [12].*

BERT[CLS] [4] differs strongly from the previously described models. It is a multi-purpose transformer based NLP model. We follow the approach from Nogueira et al. [13] and first concatenate the query and document sequences with the *[SEP]* indicator. Then, we apply a single linear layer on top of the first *[CLS]* token to produce the relevance score.

2.2 Experiment Setup

In our experiment setup, we largely follow Hofstätter et al. [6]. We use PyTorch [16] and AllenNLP [5] for the neural models and Anserini [23] to obtain the initial BM25 rankings. The BM25 baseline reaches 0.192 MRR@10 – as all neural models are significantly better, we omit it in the rest of the paper. We use the Adam optimizer and pairwise margin ranking loss with a learning rate of $1e^{-3}$ for all non-*BERT* models; for *BERT* we use a rate of $3e^{-6}$ and the "bert-base-uncased" pre-trained model[2]. We train the models with a batch size of 64; for evaluation we use a batch size of 256. We keep the defaults for the model configurations from their respective papers, except for MatchPyramid where we follow the 5-layer configuration from [6]. For the basic word embedding we use a vocabulary with a minimum collection occurrence of 5. The nature of the passage collection means we operate on fairly short text sequences: We clip the passages at 200 and the queries at 20 tokens – this only removes a modest number of outliers.

In their work, Hofstätter et al. [6] evaluate the effectiveness of the models along the re-ranking depth (i.e. how many documents are re-ranked by the neural model) – they show that a shallow re-ranking depth already saturates most queries. This insight can be employed to tune the performance of re-ranking systems further in the future. In our case study, we keep it simple by reporting the best validation MRR@10 (Mean Reciprocal Rank) results per model.

We present average timings per batch assuming a batch contains a single query with 256 re-ranking documents. We report timings from already cached batches – excluding the pre-processing and therefore reducing the considerable negative performance impact of Python as much as possible. We use a benchmark server with NVIDIA GTX 1080 TI (11GB memory) GPUs and Intel Xeon E5-2667 @ 3.20GHz CPUs. Each model is run on a single GPU.

We caution that the measurements do not reflect production ready implementations – as we directly measured PyTorch research models and we strongly believe that overall the performance can further be improved by employing more inference optimized runtimes (such as the ONNX runtime[3]) and performance optimized support code (for example non-Python code feeding data into the neural network). We would like to kick-start innovation in this direction with our paper.

---

[2]From: *https://github.com/huggingface/pytorch-pretrained-BERT*
[3]*https://github.com/microsoft/onnxruntime*

Table 1: Training performance (Triples/second includes: 2x forward, 1x loss & backward per triple), Training duration (best validation result after batch count), peak GPU memory requirement as well as the effectiveness score (MRR@10)

	Model	Triples / second	Batch count	Peak Memory	MRR
Word vectors	KNRM	5,200	44,000	2.16 GB	0.222
	C-KNRM	1,300	98,000	2.73 GB	0.261
	MatchP.	2,900	178,000	2.30 GB	0.245
	PACRR	2,900	130,000	2.21 GB	0.249
	DUET	1,900	146,000	2.47 GB	0.259
FastText	KNRM	2,300	62,000	7.34 GB	0.231
	C-KNRM	1,000	184,000	7.81 GB	0.273
	MatchP.	1,800	182,000	7.47 GB	0.254
	PACRR	1,700	100,000	7.40 GB	0.257
	DUET	1,600	182,000	7.46 GB	0.271
	BERT[CLS]	33	77,500	7.68 GB	0.347

Table 2: Re-ranking speed (256 documents per query & batch), peak GPU memory requirement and MRR@10 effectiveness of our evaluated neural IR models.

	Model	Docs / second	Time / query	GPU Memory	MRR
Word vectors	KNRM	48,000	5 ms	0.84 GB	0.222
	C-KNRM	12,000	21 ms	0.93 GB	0.261
	MatchP.	28,000	9 ms	0.97 GB	0.245
	PACRR	27,000	9 ms	0.91 GB	0.249
	DUET	14,000	18 ms	1.04 GB	0.259
FastText	KNRM	36,000	7 ms	2.59 GB	0.231
	C-KNRM	11,000	23 ms	2.68 GB	0.273
	MatchP.	23,000	11 ms	2.72 GB	0.254
	PACRR	21,000	12 ms	2.67 GB	0.257
	DUET	17,000	15 ms	2.68 GB	0.271
	BERT[CLS]	130	1,970 ms	7.29 GB	0.347

2.3 Results & Discussion

We start our observations with the training of the models as shown in Table 1. The main performance metric is the throughput of triples per second. A triple is a single training sample with a query and one relevant and one non-relevant document. The models are trained with a pairwise ranking loss, which requires two forward and a single backward pass per triple. The batch count is the number of batches with the best validation result. *KNRM* is the fastest to train, it also saturates first. *MatchPyramid* and *PACRR* exhibit similar performance. This is due to their similar architecture components (CNNs applied on a 2D match matrix). In the class of CNNs applied to higher dimensional word representations, *DUET* is slightly faster than *CONV-KNRM*, although *CONV-KNRM* is slightly more effective. In general, FastText vectors improve all models with a modest performance decrease. The peak GPU memory required in the
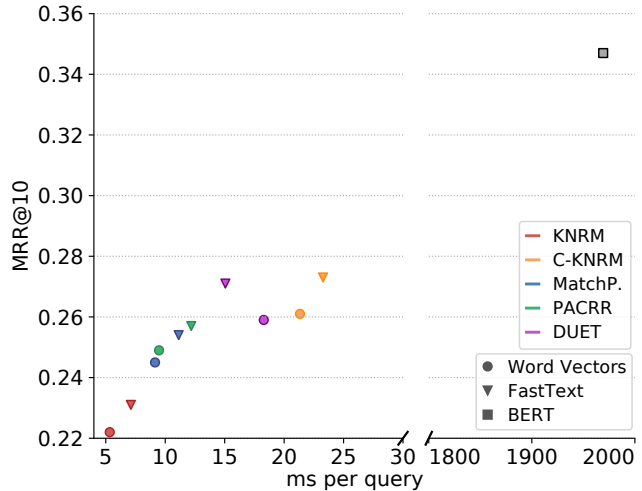

Figure 1: A comparison of performance and effectiveness
Note: the break in the x-axis indicates a large time gap

training largely depends on the encoding layer [4]. Fine-tuning the *BERT* model is much slower than all other models. It also is more challenging to fit on a GPU with limited available memory, we employed gradient accumulation to update the weights every 64 samples. We did not observe big performance differences between batch sizes.

Now we focus on the practically more important aspect: the re-ranking performance of the neural IR models. In Table 2 we report the time that the neural IR models spend to score the provided query-document pairs. The reported time only includes the model computation. This corresponds to benchmark scenario #2 (Section 3.2).

The main observation from the re-ranking performance data in Table 2 is the striking difference between *BERT* and non-*BERT* models. Both the word vector and FastText encodings have a low memory footprint and the level of performance makes them suitable for realtime re-ranking tasks. There are slight trade-offs between the models as depicted in Figure 1. The differences correspond to the training speed, discussed above. However, compared to *BERT*'s performance those differences become marginal. *BERT* offers impressive effectiveness gains at a substantial performance cost. We only evaluate a single *BERT* model, however the performance characteristics should apply to all *BERT*-based models.

We believe that the practical applicability of *BERT*-based re-ranking models is currently limited to offline scoring or domains where users are willing to accept multiple second delays in their search workflow. Future work will likely focus on the gap between the contextualized and non-contextualized models – both in terms of performance and effectiveness. Another path is to speed-up *BERT* and other transformer based models, for example with pruning [21]. Therefore, we argue that it is necessary to provide the replicability infrastructure with tools to take both performance and effectiveness dimensions into account.

---

[4]We report peak memory usage provided by PyTorch, however we observed that one requires additional GPU memory, FastText & *BERT* are not trainable on 8 GB. We believe this is due to memory fragmentation. The size of the required headroom remains an open question for future work.
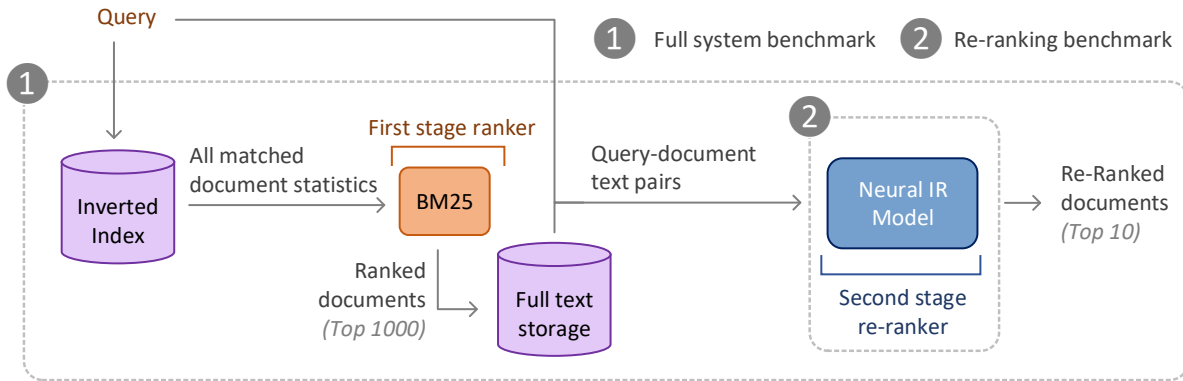
Figure 2: A simplified query workflow with re-ranking – showing the reach of our proposed performance benchmarks

3 BENCHMARK SCENARIOS

Following the observations from the case study above, we propose to systematically measure and report performance metrics as part of all replicability campaigns. Concretely, we propose to extend the OSIRRC docker-based replicability infrastructure for two additional use cases. The different measured components are depicted in Figure 2.

3.1 Full System Benchmark

Currently, most IR evaluation is conducted in batched processes – working through a set of queries at a time, as we are mostly interested in the effectiveness results. The OSIRRC specifications also contain an optional timing feature for batched retrieval[5]. While we see this as a good first step, we envision a more performance focused benchmark: A scripted "interactive" mode, where the system answers one query at a time. Here, the benchmark decides the load and is able to measure fine grained latency and throughput.

The scripted "interactive" mode needs as little overhead as possible, like a lightweight HTTP endpoint receiving the query string and returning TREC-formatted results. The execution of the benchmark needs to be conducted on the same hardware, multiple times to reduce noise.

Although we present a neural IR model case study, we do not limit this benchmark scenario to them – rather we see it as an opportunity to cover the full range of retrieval methods. For example, we are able to incorporate recall-boosting measures in the first stage retrieval such as *BERT*-based document expansion [14] or query expansion with IR-specific word embeddings [7].

Measuring the latency and throughput over a longer realistic period of a full search engine (with neural IR re-ranking component) touches many previously undeveloped components: Storing neural model input values of indexed documents, generating batches on the fly, or handling concurrency. If a neural IR model is to be deployed in production with a GPU acceleration, the issue of concurrent processing becomes important: We observed that slower models also have a higher GPU utilization, potentially creating a traffic jam on the GPU, that in turn would increase the needed infrastructure cost for the same amount of users.

3.2 Re-ranking Benchmark

The neural IR field is receiving considerable attention and has a growing community. In our opinion, the community is in need of a more structured evaluation – both for performance and effectiveness. We now propose a benchmark, which aims to deliver on both dimensions.

The re-ranking benchmark focuses on the innermost component of neural IR models: the scoring of query-document tuples. We provide the re-ranking candidate list and the neural IR model scores the tuples. Many of the existing neural IR models follow this pattern and can therefore easily be swapped and compared with each other – also on public leaderboards, such as the MS MARCO leaderboard. This static scenario provides a coherent way of evaluating neural IR re-ranking models. It helps to mitigate differences in the setup of various research groups.

4 CONCLUSION

The OSIRRC docker-based IR replicability infrastructure presents an opportunity to incorporate performance benchmarks. As an example for the need of a broader view of the community, we show in a case study the trade-off between performance and effectiveness of neural IR models, especially for recent *BERT* based models. As a result, we propose two different performance-focused benchmarks to be incorporated in the infrastructure going forward. We look forward to working with the community on these issues.

REFERENCES

[1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew Mcnamara, Bhaskar Mitra, and Tri Nguyen. 2016. MS MARCO : A Human Generated MAchine Reading COmprehension Dataset. In *Proc. of NIPS*.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Tr. of the ACL* 5 (2017).

[3] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *Proc. of WSDM*.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, et al. 2017. AllenNLP: A Deep Semantic Natural Language Processing Platform. arXiv:arXiv:1803.07640

[6] Sebastian Hofstätter, Navid Rekabsaz, Carsten Eickhoff, and Allan Hanbury. 2019. On the Effect of Low-Frequency Terms on Neural-IR Models. In *Proc. of SIGIR*.

---

[5]See: https://github.com/osirrc/jig

[7] Sebastian Hofstätter, Navid Rekabsaz, Mihai Lupu, Carsten Eickhoff, and Allan Hanbury. 2019. Enriching Word Embeddings for Patent Retrieval with Global Context. In *Proc. of ECIR*.

[8] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, et al. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proc. of the IEEE-CVPR*.

[9] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. 2017. PACRR: A Position-Aware Neural IR Model for Relevance Matching. In *Proc. of EMNLP*.

[10] Oscar Jimenez-del Toro, Henning Müller, Markus Krenn, et al. 2016. Cloud-based evaluation of anatomical structure segmentation and landmark detection algorithms: VISCERAL anatomy benchmarks. *IEEE trans. on Med. Imaging* (2016).

[11] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *SIGIR*.

[12] Bhaskar Mitra and Nick Craswell. 2019. An Updated Duet Model for Passage Re-ranking. *arXiv preprint arXiv:1903.07666* (2019).

[13] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[14] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. *arXiv preprint arXiv:1904.08375* (2019).

[15] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching as Image Recognition. In *Proc of. AAAI*.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, et al. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[17] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc of EMNLP*.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proc. of the IEEE-CVPR*.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proc. of NIPS*.

[20] Jaime Teevan, Kevyn Collins-Thompson, Ryen W White, Susan T Dumais, and Yubin Kim. 2013. Slow search: Information retrieval without time constraints. In *Proc. of the Symposium on HCI and IR*.

[21] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *arXiv preprint arXiv:1905.09418* (2019).

[22] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proc. of SIGIR*.

[23] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of Lucene for information retrieval research. In *Proc. of SIGIR*.