Overview of the 2019 Open-Source IR Replicability Challenge (OSIRRC 2019)

Ryan Clancy,[1] Nicola Ferro,[2] Claudia Hauff,[3] Jimmy Lin,[1] Tetsuya Sakai,[4] Ze Zhong Wu[1]
[1] University of Waterloo [2] University of Padua [3] TU Delft [4] Waseda University

ABSTRACT

The Open-Source IR Replicability Challenge (OSIRRC 2019), organized as a workshop at SIGIR 2019, aims to improve the replicability of *ad hoc* retrieval experiments in information retrieval by gathering a community of researchers to jointly develop a common Docker specification and build Docker images that encapsulate a diversity of systems and retrieval models. We articulate the goals of this workshop and describe the "jig" that encodes the Docker specification. In total, 13 teams from around the world submitted 17 images, most of which were designed to produce retrieval runs for the TREC 2004 Robust Track test collection. This exercise demonstrates the feasibility of orchestrating large, community-based replication experiments with Docker technology. We envision OSIRRC becoming an ongoing community-wide effort to ensure experimental replicability and sustained progress on standard test collections.

1 INTRODUCTION

The importance of repeatability, replicability, and reproducibility is broadly recognized in the computational sciences, both in supporting desirable scientific methodology as well as sustaining empirical progress. The Open-Source IR Replicability Challenge (OSIRRC 2019), organized as a workshop at SIGIR 2019, aims to improve the replicability of *ad hoc* retrieval experiments in information retrieval by building community consensus around a common technical specification, with reference implementations. This overview paper is an extended version of an abstract that appears in the SIGIR proceedings.

In order to precisely articulate the goals of this workshop, it is first necessary to establish common terminology. We use the above terms in the same manner as recent ACM guidelines pertaining to artifact review and badging:[1]

- *Repeatability* (same team, same experimental setup): a researcher can reliably repeat her own computation.
- *Replicability* (different team, same experimental setup): an independent group can obtain the same result using the authors' own artifacts.
- *Reproducibility* (different team, different experimental setup): an independent group can obtain the same result using artifacts which they develop completely independently.

This workshop tackles the replicability challenge for *ad hoc* document retrieval, with three explicit goals:

(1) Develop a common Docker specification to support images that capture systems performing *ad hoc* retrieval experiments on standard test collections. The solution that we have developed is known as "the jig".
(2) Build a curated library of Docker images that work with the jig to capture a diversity of systems and retrieval models.
(3) Explore the possibility of broadening our efforts to include additional tasks, diverse evaluation methodologies, and other benchmarking initiatives.

Trivially, by supporting replicability, our proposed solution enables repeatability as well (which, as a recent case study has shown [14], is not as easy as one might imagine). It is *not* our goal to directly address reproducibility, although we do see our efforts as an important stepping stone.

We hope that the fruits of this workshop can fuel empirical progress in *ad hoc* retrieval by providing competitive baselines that are easily replicable. The "prototypical" research paper of this mold proposes an innovation and demonstrates its value by comparing against one or more baselines. The often-cited meta-analysis of Armstrong et al. [2] from a decade ago showed that researchers compare against weak baselines, and a recent study by Yang et al. [13] revealed that, a decade later, the situation has not improved much—researchers are *still* comparing against weak baselines. Lin [9] discussed social aspects of why this persists, but there are genuine technical barriers as well. The growing complexity of modern retrieval techniques, especially neural models that are sensitive to hyperparameters and other details of the training regime, poses challenges for researchers who wish to demonstrate that their proposed innovation improves upon a particular method. Solutions that address replicability facilitate in-depth comparisons between existing and proposed approaches, potentially leading to more insightful analyses and accelerating advances.

Overall, we are pleased with progress towards the first two goals of the workshop. A total of 17 Docker images, involving 13 different teams from around the world, were submitted for evaluation, comprising the OSIRRC 2019 "image library". These images collectively generated 49 replicable runs for the TREC 2004 Robust Track test collection, 12 replicable runs for the TREC 2017 Common Core Track test collection, and 19 replicable runs for the TREC 2018 Common Core Track test collection. With respect to the third goal, this paper offers our future vision—but its broader adoption by the community at large remains to be seen.

2 BACKGROUND

There has been much discussion about reproducibility in the sciences, with most scientists agreeing that the situation can be characterized as a crisis [3]. We lack the space to provide a comprehensive review of relevant literature in the medical, natural, and behavioral sciences. Within the computational sciences, to which at least a large portion of information retrieval research belongs, there have been many studies and proposed solutions, for example, a

[1] https://www.acm.org/publications/policies/artifact-review-badging

recent Dagstuhl seminar [7]. Here, we focus on summarizing the immediate predecessor of this workshop.

Our workshop was conceived as the next iteration of the Open-Source IR Reproducibility Challenge (OSIRRC), organized as part of the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR) [1]. This event in turn traces its roots back to a series of workshops focused on open-source IR systems, which is widely understood as an important component of reproducibility. The Open-Source IR Reproducibility Challenge[2] brought together developers of open-source search engines to provide replicable baselines of their systems in a common environment on Amazon EC2. The product is a repository that contains all code necessary to generate *ad hoc* retrieval baselines, such that with a single script, anyone with a copy of the collection can replicate the submitted runs. Developers from seven different systems contributed to the evaluation, which was conducted on the GOV2 collection. The details of their experience are captured in an ECIR 2016 paper [10].

In OSIRRC 2019, we aim to address two shortcomings with the previous exercise as a concrete step in moving the field forward. From the technical perspective, the RIGOR 2015 participants developed scripts in a shared VM environment, and while this was sufficient to support cross-system comparisons at the time, the scripts were not sufficiently constrained, and the entire setup suffered from portability and isolation issues. Thus, it would have been difficult for others to reuse the infrastructure to replicate the results—in other words, the replicability experiments themselves were difficult to replicate. We believe that Docker, which is a popular standard for containerization, offers a potential solution to these technical challenges.

Another limitation of the previous exercise was its focus on "bag of words" baselines, and while some participants did submit systems that exploited richer models (e.g., term dependence models and pseudo-relevance feedback), there was insufficient diversity in the retrieval models examined. Primarily due to these issues, the exercise has received less follow-up and uptake than the organizers had originally hoped.

3 DOCKER AND "THE JIG"

From a technical perspective, our efforts are built around Docker, a widely-adopted Linux-centric technology for delivering software in lightweight packages called containers. The Docker Engine hosts one or more of these containers on physical machines and manages their lifecycle. One key feature of Docker is that all containers run on a single operating system kernel; isolation is handled by Linux kernel features such as cgroups and kernel namespaces. This makes containers far more lightweight than virtual machines, and hence easier to manipulate. Containers are created from *images*, which are typically built by importing base images (for example, capturing a specific software distribution) and then overlaying custom code. The images themselves can be manipulated, combined, and modified as first-class citizens in a broad ecosystem. For example, a group can overlay several existing images from public sources, add in its own code, and in turn publish the resulting image to be further used by others.

3.1 General Design

As defined by the Merriam-Webster dictionary, a jig is "a device used to maintain mechanically the correct positional relationship between a piece of work and the tool or between parts of work during assembly". The central activity of this workshop revolved around the co-design and co-implementation of a jig and Docker images that work with the jig for *ad hoc* retrieval. Of course, in our context, the relationship is computational instead of mechanical.

Shortly after the acceptance of the workshop proposal at SIGIR 2019, we issued a call for participants who were interested in contributing Docker images to our effort; the jig was designed with the input of these participants. In other words, the jig and the images co-evolved with feedback from members of the community. The code of the jig is open source and available on GitHub.[3]

Our central idea is that each image would expose a number of "hooks" that correspond to a point in the prototypical lifecycle of an *ad hoc* retrieval experiment: for example, indexing a collection, running a batch of queries, etc. These hooks then tie into code that captures whatever retrieval model a particular researcher wishes to package in the image—for example, a search engine implemented in Java or C++. The jig is responsible for triggering the hooks in each image in a particular sequence according to a predefined lifecycle model, e.g., first index the collection, then run a batch of queries, finally evaluate the results. We have further built tooling that applies the jig to multiple images to facilitate large-scale experiments. More details about the jig are provided in the next section, but first we overview a few design decisions.

Quite deliberately, the current jig does not make any demands about the transparency of a particular image. For example, the search hook can run an executable whose source code is not publicly available. Such an image, while demonstrating replicability, would not allow other researchers to inspect the inner workings of a particular retrieval method. While such images are not forbidden in our design, they are obviously less desirable than images based on open code. In practice, however, we anticipate that most images will be based on open-source code.

One technical design choice that we have grappled with is how to get data "into" and "out of" a container. To be more concrete, for *ad hoc* retrieval the container needs access to the document collection and the topics. The jig also needs to be able to obtain the run files generated by the image for evaluation. Generically, there are three options for feeding data to an image: first, the data can be part of the image itself; second, the data can be fetched from a remote location by the image (e.g., via `curl`, `wget`, or some other network transfer mechanism); third, the jig could mount an external data directory that the container has access to. The first two approaches are problematic for our use case: images need to be shareable, or resources need to be placed at a publicly-accessible location online. This is not permissible for document collections where researchers are required to sign license agreements before using. Furthermore, both approaches do not allow the possibility of testing on blind held-out data. We ultimately opted for the third

[2]Note that the exercise is more accurately characterized as replicability and not reproducibility; the event predated ACM's standardization of terminology.

[3]https://github.com/osirrc/jig

approach: the jig mounts a (read-only) data directory that makes the document collection available at a known location, as part of the contract between the jig and the image (and similarly for topics). A separate directory that is writable serves as the mechanism for the jig to gather output runs from the image for evaluation. This method makes it possible for images to be tested on blind held-out documents and topics, as long as the formats have been agreed to in advance.

Finally, any evaluation exercise needs to define the test collection. We decided to focus on newswire test collections because their smaller sizes support a shorter iteration and debug cycle (compared to, for example, larger web collections). In particular, we asked participants to focus on the TREC 2004 Robust Track test collection, in part because of its long history: a recent large-scale literature meta-analysis comprising over one hundred papers [13] provides a rich context to support historical comparisons.

Participants were also asked to accommodate the following two (more recent) test collections if time allowed:

- TREC 2017 Common Core Track, on the New York Times Annotated Corpus.
- TREC 2018 Common Core Track, on the Washington Post Corpus.

Finally, a "reach" goal was to support existing web test collections (e.g., GOV2 and ClueWeb). Although a few submitted images do support one or more of these collections, no formal evaluation was conducted on them.

3.2 Implementation Details

In this section we provide a more detailed technical description of the jig. Note, however, that the jig is continuously evolving as we gather more image contributions and learn about our design shortcomings. We invite interested readers to consult our code repository for the latest details and design revisions. To be clear, we describe v0.1.1 of the jig, which was deployed for the evaluation.

The jig is implemented in Python and communicates with the Docker Engine via the Docker SDK for Python.[4] In the current specification, each hook corresponds to a script in the image that has a specific name, resides at a fixed location, and obeys a specified contract dictating its behavior. Each script can invoke its own interpreter: common implementations include bash and Python. Thus, via these scripts, the image has freedom to invoke arbitrary code. In the common case, the hooks invoke features of an existing open-source search engine packaged in the image.

From the perspective of a user who is attempting to replicate results using an image, two commands are available: one for preparation (the `prepare` phase) and another for actually performing the *ad hoc* retrieval run (the `search` phase). The jig handles the execution lifecycle, from downloading the image to evaluating run files using `trec_eval`. This is shown in Figure 1 as a timeline in the canonical lifecycle, with the jig on the left and the Docker image on the right. The two phases are described in detail below.

During the `prepare` phase, the user issues a command specifying an image's repository (i.e., name) and tag (i.e., version) along with a list of collections to index. As part of the contract between the jig and an image, the jig mounts the document collections and makes

[4] https://docker-py.readthedocs.io/en/stable/

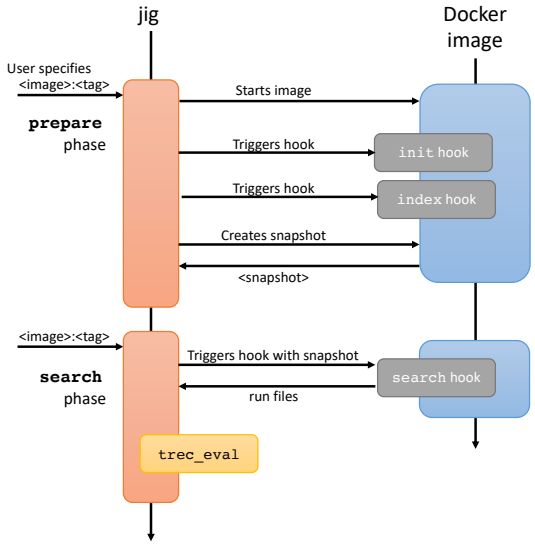

Figure 1: Interactions between the jig and the Docker image in the canonical evaluation flow.

them readable by the image for indexing (see discussion in the previous section). The jig triggers two hooks in the image:

- First, the `init` hook is executed. This hook is meant for actions such as downloading artifacts, cloning repositories and compiling source code, or downloading external resources (e.g., a knowledge graph). Alternatively, these steps can be encoded directly in the image itself, thus making `init` a no-op. These two mechanisms largely lead to the same end result, and so it is mostly a matter of preference for the image developer.
- Next, the `index` hook is executed. The jig passes in a JSON string containing information such as the collection name, path, format, etc. required for indexing. The image manages its own index, which is not directly visible to the jig.

After indexing has completed, the jig takes a snapshot of the image via a Docker commit. This is useful as indexing generally takes longer than a retrieval run, and this design allows multiple runs (at different times) to be performed using the same index.

During the `search` phase, the user issues a command specifying an image's repository (i.e., name) and tag (i.e., version), the collection to search, and a number of auxiliary parameters such as the topics file, qrels file, and output directory. This hook is meant to perform the actual *ad hoc* retrieval runs, after which the jig evaluates the output with `trec_eval`. Just as in the `index` hook, relevant parameters are encoded in JSON. The image places run files in the `/output` directory, which is mapped back to the host; this allows the jig to retrieve the run files for evaluation.

In addition to the two main hooks for *ad hoc* retrieval experiments, the jig also supports additional hooks for added functionality (not shown in Figure 1). The first of these is the `interact` hook that allows a user to interactively explore an image in the state that has been captured via a snapshot, after the execution of the `index` hook. This allows, for example, the user to "enter" an interactive shell in the image (via standard Docker commands), and allows

the user to explore the inner workings of an image. The hook also allows users to interact with services that a container may choose to expose, such as an interactive search interface, or even Jupyter notebooks. With the `interact` hook, the container is kept alive in the foreground, unlike the other hooks which exit immediately once execution has finished.

Finally, images may also implement a `train` hook, enabling an image to train a retrieval model, tune hyper-parameters, etc. after the `index` hook has been executed. The `train` hook allows the user to specify training and test splits for a set of topics, along with a model directory for storing the model. This output directory is mapped back to the host and can be passed to the `search` hook for use during retrieval. Currently, training is limited to the CPU, although progress has been made to support GPU-based training.

In the current design, the jig runs one image at a time, but additional tooling around the jig includes a script that further automates all interactions with an image so that experiments can be run end to end with minimal human supervision. This script creates a virtual machine in the cloud (currently, Microsoft Azure), installs Docker engine and associated dependencies, and then runs the image using the jig. All output is then captured for archival purposes.

4 SUBMITTED IMAGES AND RESULTS

Although we envision OSIRRC to be an ongoing effort, the reality of a physical SIGIR workshop meant that it was necessary to impose an arbitrary deadline at which to "freeze" image development. This occurred at the end of June, 2019. At that point in time, we received 17 images by 13 different teams, listed alphabetically as follows:

- Anserini (University of Waterloo)
- Anserini-bm25prf (Waseda University)
- ATIRE (University of Otago)
- Birch (University of Waterloo)
- Elastirini (University of Waterloo)
- EntityRetrieval (Ryerson University)
- Galago (University of Massachusetts)
- ielab (University of Queensland)
- Indri (TU Delft)
- IRC-CENTRE2019 (Technische Hochschule Köln)
- JASS (University of Otago)
- JASSv2 (University of Otago)
- NVSM (University of Padua)
- OldDog (Radboud University)
- PISA (New York University and RMIT University)
- Solrini (University of Waterloo)
- Terrier (TU Delft and University of Glasgow)

All except for two images were designed to replicate runs for the TREC 2004 Robust Track test collection, which was the primary target for the exercise. The EntityRetrieval image was designed to perform entity retrieval (as opposed to *ad hoc* retrieval). The IRC-CENTRE2019 image packages a submission to the CENTRE reproducibility effort,[5] which targets a specific set of runs from the TREC 2017 Common Core Track. A number of images also support the Common Core Track test collections from TREC 2017 and 2018. Finally, a few images also provide support for the GOV2 and ClueWeb test collections, although these were not evaluated.

Following the deadline for submitting images, the organizers ran all images "from scratch" with v0.1.1 of the jig and the latest release of each participant's image. Using our script (see Section 3.2), each image was executed sequentially on a virtual machine instance in the Microsoft Azure cloud. Note that it would have been possible to speed up the experiments by running the images in parallel, each on its own virtual machine instance, but this was not done. We used the instance type `Standard_D64s_v3`, which according to Azure documentation is either based on the 2.4 GHz Intel Xeon E5-2673 v3 (Haswell) processor or 2.3 GHz Intel Xeon E5-2673 v4 (Broadwell) processor. Since we have no direct control over the physical hardware, it is only meaningful to compare efficiency (i.e., performance metrics such as query latency) across different images running on the same virtual machine instance. Nevertheless, our evaluations focused solely on retrieval effectiveness. This is a shortcoming, since a number of images packaged search engines that emphasize query evaluation efficiency.

The results of running the jig on the submitted images comprise the "official" OSIRRC 2019 image library, and is available on GitHub.[6] We have captured all log output, run files, as well as `trec_eval` output. These results are summarized below.

For the TREC 2004 Robust Track test collection, 13 images generated a total of 49 runs, the results of which are shown in Table 1; the specific version of the image is noted. Effectiveness is measured using standard rank retrieval metrics: average precision (AP), precision at rank cutoff 30 (P30), and NDCG at rank cutoff 20 (NDCG@20). The table does not include runs from the following images: Solrini and Elastirini (which are identical to Anserini runs), EntityRetrieval (where relevance judgments are not available since it was designed for a different task), and IRC-CENTRE2019 (which was not designed to produce results for this test collection).

As the primary goal of this workshop is to build community, infrastructure, and consensus, we deliberately attempt to minimize direct comparisons of run effectiveness in the presentation: runs are grouped by image, and the image themselves are sorted alphabetically. Nevertheless, a few important caveats are necessary for proper interpretation of the results: Most runs perform no parameter tuning, although at least one implicitly encodes cross-validation results (e.g., Birch). Also, runs might use different parts of the complete topic: the "title", "description", and "narrative" (as well as various combinations). For details, we invite the reader to consult the overview paper by each participating team.

We see that the submitted images generate runs that use a diverse set of retrieval models, including query expansion and pseudo-relevance feedback (Anserini, Anserini-bm25prf, Indri, Terrier), term proximity (Indri and Terrier), conjunctive query processing (OldDog), and neural ranking models (Birch and NVSM). Several images package open-source search engines that are primarily focused on efficiency (ATIRE, JASS, JASSv2, PISA). Although we concede that there is an under-representation of neural approaches, relative to the amount of interest in the community at present, there are undoubtedly replication challenges with neural ranking models, particularly with their training regimes. Nevertheless, we are pleased with the range of systems and retrieval models that are represented in these images.

[5] http://www.centre-eval.org/

[6] https://github.com/osirrc/osirrc2019-library

Results from the TREC 2017 Common Core Track test collection are shown in Table 2. On this test collection, we have 12 runs from 6 images. Results from the TREC 2018 Common Core Track test collection are shown in Table 3: there are 19 runs from 4 images.

5 FUTURE VISION AND ONGOING WORK

Our efforts complement other concurrent activities in the community. SIGIR has established a task force to implement ACM's policy on artifact review and badging [5], and our efforts can be viewed as a technical feasibility study. This workshop also complements the recent CENTRE evaluation tasks jointly run at CLEF, NTCIR, and TREC [6, 11]. One of the goals of CENTRE is to define appropriate measures to determine whether and to what extent replicability and reproducibility have been achieved, while our efforts focus on how these properties can be demonstrated technically. Thus, the jig can provide the means to achieve CENTRE goals. Given fortuitous alignment in schedules, participants of CENTRE@CLEF2019 [4] were encouraged to participate in our workshop, and this in fact led to the contribution of the IRC-CENTRE2019 image.

From the technical perspective, we see two major shortcomings of the current jig implementation. First, the training hook is not as well-developed as we would have liked. Second, the jig lacks GPU support. Both will be remedied in a future iteration.

We have proposed and prototyped a technical solution to the replicability challenge specifically for the SIGIR community, but the changes we envision will not occur without a corresponding cultural shift. Sustained, cumulative empirical progress will only be made if researchers use our tools in their evaluations, and this will only be possible if images for the comparison conditions are available. This means that the community needs to adopt the norm of associating research papers with source code for replicating results in those papers. However, as Voorhees et al. [12] reported, having a link to a repository in a paper is far from sufficient. The jig provides the tools to package *ad hoc* retrieval experiments in a standard way, but these tools are useless without broad adoption. The incentive structures of academic publishing need to adapt to encourage such behavior, but unfortunately this is beyond the scope of our workshop.

Given appropriate extensions, we believe that the jig can be augmented to accommodate a range of batch retrieval tasks. One important future direction is to add support for tasks beyond batch retrieval, for example, to support interactive retrieval (with real or simulated user input) and evaluations on private and other sensitive data. Moreover, our effort represents a first systematic attempt to embody the Evaluation-as-a-Service paradigm [8] via Docker containers. We believe that there are many possible paths forward building on the ideas presented here.

Finally, we view our efforts as a stepping stone toward reproducibility, and beyond that, generalizability. While these two important desiderata are not explicit goals of our workshop, we note that the jig itself can provide the technical vehicle for delivering reproducibility and generalizability. Some researchers would want to package their own results in a Docker image. However, there is nothing that would prevent researchers from reproducing another team's results, that is then captured in a Docker image conforming to our specifications. This would demonstrate reproducibility as well as replicability of those reproducibility efforts. The jig also supports mechanisms for evaluations on document collections and information needs beyond those that an image was originally designed for. This aligns with intuitive notions of what it means for a technique to be generalizable.

Overall, we believe that our efforts have moved the field of information retrieval forward both in terms of supporting "good science" as well as sustained, cumulative empirical progress. This work shows that it is indeed possible to coordinate a large, community-wide replication exercise in *ad hoc* retrieval, and that Docker provides a workable foundation for a common interface and lifecycle specification. We invite the broader community to join our efforts!

6 ACKNOWLEDGEMENTS

We would like to thank all the participant who contributed Docker images to the workshop. This exercise would not have been possible without their efforts. Additional thanks to Microsoft for providing credits on the Azure cloud.

REFERENCES

[1] Jaime Arguello, Matt Crane, Fernando Diaz, Jimmy Lin, and Andrew Trotman. 2015. Report on the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum* 49, 2 (2015), 107–116.

[2] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Improvements That Don't Add Up: Ad-Hoc Retrieval Results Since 1998. In *Proceedings of the 18th International Conference on Information and Knowledge Management (CIKM 2009)*. Hong Kong, China, 601–610.

[3] Monya Baker. 2016. Is There a Reproducibility Crisis? *Nature* 533 (2016), 452–454.

[4] Nicola Ferro, Norbert Fuhr, Maria Maistro, Tetsuya Sakai, and Ian Soboroff. 2019. Overview of CENTRE@CLEF 2019: Sequel in the Systematic Reproducibility Realm. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Tenth International Conference of the CLEF Association (CLEF 2019)*. Lugano, Switzerland.

[5] Nicola Ferro and Diane Kelly. 2018. SIGIR Initiative to Implement ACM Artifact Review and Badging. *SIGIR Forum* 52, 1 (2018), 4–10.

[6] Nicola Ferro, Maria Maistro, Tetsuya Sakai, and Ian Soboroff. 2018. Overview of CENTRE@CLEF 2018: A First Tale in the Systematic Reproducibility Realm. In *Proceedings of the Ninth International Conference of the CLEF Association (CLEF 2018)*. Avignon, France, 239–246.

[7] Juliana Freire, Norbert Fuhr, and Andreas Rauber (Eds.). 2016. *Report from Dagstuhl Seminar 16041: Reproducibility of Data-Oriented Experiments in e-Science*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Germany.

[8] Frank Hopfgartner, Allan Hanbury, Henning Müller, Ivan Eggel, Krisztian Balog, Torben Brodt, Gordon V. Cormack, Jimmy Lin, Jayashree Kalpathy-Cramer, Noriko Kando, Makoto P. Kato, Anastasia Krithara, Tim Gollub, Martin Potthast, Evelyne Viega, and Simon Mercer. 2018. Evaluation-as-a-Service for the Computational Sciences: Overview and Outlook. *ACM Journal of Data and Information Quality (JDIQ)* 10, 4 (November 2018), 15:1–15:32.

[9] Jimmy Lin. 2018. The Neural Hype and Comparisons Against Weak Baselines. *SIGIR Forum* 52, 2 (2018), 40–51.

[10] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *Proceedings of the 38th European Conference on Information Retrieval (ECIR 2016)*. Padua, Italy, 408–420.

[11] Tetsuya Sakai, Nicola Ferro, Ian Soboroff, Zhaohao Zeng, Peng Xiao, and Maria Maistro. 2019. Overview of the NTCIR-14 CENTRE Task. In *Proceedings of the 14th NTCIR Conference on Evaluation of Information Access Technologies*. Tokyo, Japan.

[12] Ellen M. Voorhees, Shahzad Rajput, and Ian Soboroff. 2016. Promoting Repeatability Through Open Runs. In *Proceedings of the 7th International Workshop on Evaluating Information Access (EVIA 2016)*. Tokyo, Japan, 17–20.

[13] 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 *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*. Paris, France.

[14] Ruifan Yu, Yuhao Xie, and Jimmy Lin. 2018. H_2oloo at TREC 2018: Cross-Collection Relevance Transfer for the Common Core Track. In *Proceedings of the Twenty-Seventh Text REtrieval Conference (TREC 2018)*. Gaithersburg, Maryland.

Image	Version	Run	AP	P30	NDCG@20
Anserini	v0.1.1	bm25	0.2531	0.3102	0.4240
Anserini	v0.1.1	bm25.rm3	0.2903	0.3365	0.4407
Anserini	v0.1.1	bm25.ax	0.2895	0.3333	0.4357
Anserini	v0.1.1	ql	0.2467	0.3079	0.4113
Anserini	v0.1.1	ql.rm3	0.2747	0.3232	0.4269
Anserini	v0.1.1	ql.ax	0.2774	0.3229	0.4223
Anserini-bm25prf	v0.2.2	b=0.20_bm25_bm25prf	0.2916	0.3396	0.4419
Anserini-bm25prf	v0.2.2	b=0.40_bm25_bm25prf	0.2928	0.3438	0.4418
ATIRE	v0.1.1	ANT_r4_100_percent.BM25+.s-stem.RF	0.2184	0.3199	0.4211
Birch	v0.1.0	mb_2cv.cv.a	0.3241	0.3756	0.4722
Birch	v0.1.0	mb_2cv.cv.ab	0.3240	0.3756	0.4720
Birch	v0.1.0	mb_2cv.cv.abc	0.3244	0.3767	0.4738
Birch	v0.1.0	mb_5cv.cv.a	0.3266	0.3783	0.4769
Birch	v0.1.0	mb_5cv.cv.ab	0.3278	0.3795	0.4817
Birch	v0.1.0	mb_5cv.cv.abc	0.3278	0.3790	0.4831
Birch	v0.1.0	qa_2cv.cv.a	0.3014	0.3507	0.4469
Birch	v0.1.0	qa_2cv.cv.ab	0.3003	0.3494	0.4475
Birch	v0.1.0	qa_2cv.cv.abc	0.3003	0.3494	0.4475
Birch	v0.1.0	qa_5cv.cv.a	0.3102	0.3574	0.4628
Birch	v0.1.0	qa_5cv.cv.ab	0.3090	0.3577	0.4615
Birch	v0.1.0	qa_5cv.cv.abc	0.3090	0.3577	0.4614
Galago	v0.0.2	output_robust04	0.1948	0.2659	0.3732
ielab	v0.0.1	robust04-1000	0.1826	0.2605	0.3477
Indri	v0.2.1	bm25.title	0.2338	0.2995	0.4041
Indri	v0.2.1	bm25.title.prf	0.2563	0.3041	0.3995
Indri	v0.2.1	bm25.title+desc	0.2702	0.3274	0.4517
Indri	v0.2.1	bm25.title+desc.prf.sd	0.2971	0.3562	0.4448
Indri	v0.2.1	dir1000.title	0.2499	0.3100	0.4201
Indri	v0.2.1	dir1000.title.sd	0.2547	0.3146	0.4232
Indri	v0.2.1	dir1000.title.prf	0.2812	0.3248	0.4276
Indri	v0.2.1	dir1000.title.prf.sd	0.2855	0.3295	0.4298
Indri	v0.2.1	dir1000.desc	0.2023	0.2581	0.3635
Indri	v0.2.1	jm0.5.title	0.2242	0.2839	0.3689
JASS	v0.1.1	JASS_r4_10_percent	0.1984	0.2991	0.4055
JASSv2	v0.1.1	JASSv2_10	0.1984	0.2991	0.4055
NVSM	v0.1.0	robust04_test_topics_run	0.1415	0.2197	0.2757
OldDog	v1.0.0	bm25.robust04.con	0.1736	0.2526	0.3619
OldDog	v1.0.0	bm25.robust04.dis	0.2434	0.2985	0.4002
PISA	v0.1.3	robust04-1000	0.2534	0.3120	0.4221
Terrier	v0.1.7	bm25	0.2363	0.2977	0.4049
Terrier	v0.1.7	bm25_qe	0.2762	0.3281	0.4332
Terrier	v0.1.7	bm25_prox	0.2404	0.3033	0.4082
Terrier	v0.1.7	bm25_prox_qe	0.2781	0.3288	0.4307
Terrier	v0.1.7	dph	0.2479	0.3129	0.4198
Terrier	v0.1.7	dph_qe	0.2821	0.3369	0.4425
Terrier	v0.1.7	dph_prox	0.2501	0.3166	0.4206
Terrier	v0.1.7	dph_prox_qe	0.2869	0.3376	0.4435
Terrier	v0.1.7	pl2	0.2241	0.2918	0.3948
Terrier	v0.1.7	pl2_qe	0.2538	0.3126	0.4163

Table 1: Results on the TREC 2004 Robust Track test collection.

Image	Version	Run	AP	P30	NDCG@20
Anserini	v0.1.1	bm25	0.2087	0.4293	0.3877
Anserini	v0.1.1	bm25.rm3	0.2823	0.5093	0.4467
Anserini	v0.1.1	bm25.ax	0.2787	0.4980	0.4450
Anserini	v0.1.1	ql	0.2032	0.4467	0.3958
Anserini	v0.1.1	ql.rm3	0.2606	0.4827	0.4226
Anserini	v0.1.1	ql.ax	0.2613	0.4953	0.4429
ATIRE	v0.1.1	ANT_c17_100_percent	0.1436	0.4087	0.3742
IRC-CENTRE2019	v0.1.3	wcrobust04	0.2971	0.5613	0.5143
IRC-CENTRE2019	v0.1.3	wcrobust0405	0.3539	0.6347	0.5821
JASS	v0.1.1	JASS_c17_10_percent	0.1415	0.4080	0.3711
JASSv2	v0.1.1	JASSv2_c17_10	0.1415	0.4080	0.3711
PISA	v0.1.3	core17-1000	0.2078	0.4260	0.3898

Table 2: Results on the TREC 2017 Common Core Track test collection.

Image	Version	Run	AP	P30	NDCG@20
Anserini	v0.1.1	bm25	0.2495	0.3567	0.4100
Anserini	v0.1.1	bm25.ax	0.2920	0.4027	0.4342
Anserini	v0.1.1	bm25.rm3	0.3136	0.4200	0.4604
Anserini	v0.1.1	ql	0.2526	0.3653	0.4204
Anserini	v0.1.1	ql.ax	0.2966	0.4060	0.4303
Anserini	v0.1.1	ql.rm3	0.3073	0.4000	0.4366
OldDog	v1.0.0	bm25.con	0.1802	0.3167	0.3650
OldDog	v1.0.0	bm25.dis	0.2381	0.3313	0.3706
PISA	v0.1.3	core18-1000	0.2384	0.3500	0.3927
Terrier	v0.1.7	bm25	0.2326	0.3367	0.3800
Terrier	v0.1.7	bm25_qe	0.2975	0.4040	0.4290
Terrier	v0.1.7	bm25_prox	0.2369	0.3447	0.3954
Terrier	v0.1.7	bm25_prox_qe	0.2960	0.4067	0.4318
Terrier	v0.1.7	dph	0.2427	0.3633	0.4022
Terrier	v0.1.7	dph_qe	0.3055	0.4153	0.4369
Terrier	v0.1.7	dph_prox	0.2428	0.3673	0.4140
Terrier	v0.1.7	dph_prox_qe	0.3035	0.4167	0.4462
Terrier	v0.1.7	pl2	0.2225	0.3227	0.3636
Terrier	v0.1.7	pl2_qe	0.2787	0.3933	0.3975

Table 3: Results on the TREC 2018 Common Core Track test collection.