A Docker-Based Replicability Study of a Neural Information Retrieval Model

Nicola Ferro, Stefano Marchesin, Alberto Purpura, Gianmaria Silvello
Department of Information Engineering
University of Padua, Italy
{ferro,marches1,purpuraa,silvello}@dei.unipd.it

ABSTRACT

In this work, we propose a Docker image architecture for the replicability of *Neural IR (NeuIR)* models. We also share two self-contained Docker images to run the *Neural Vector Space Model (NVSM)* [22], an unsupervised NeuIR model. The first image we share (`nvsm_cpu`) can run on most machines and relies only on CPU to perform the required computations. The second image we share (`nvsm_gpu`) relies instead on the *Graphics Processing Unit (GPU)* of the host machine, when available, to perform computationally intensive tasks, such as the training of the NVSM model. Furthermore, we discuss some insights on the engineering challenges we encountered to obtain deterministic and consistent results from NeuIR models, relying on `TensorFlow` within Docker. We also provide an in-depth evaluation of the differences between the runs obtained with the shared images. The differences are due to the usage within Docker of `TensorFlow` and CUDA libraries – whose inherent randomness alter, under certain circumstances, the relative order of documents in rankings.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Retrieval models and ranking**; **Evaluation of retrieval results**; *Retrieval models and ranking*; • **Computing methodologies** → *Unsupervised learning*.

KEYWORDS

Docker, Neural Information Retrieval, Replicability, Reproducibility

1 INTRODUCTION

Following some recent efforts on reproducibility, like the CENTRE evaluations at CLEF [7, 9], NTCIR [19] and TREC [20], or the SIGIR task force to implement ACM's policy on artifact review and badging [8], the *Open-Source IR Replicability Challenge at SIGIR 2019 (OSIRRC 2019)* aims at addressing the replicability issue in *ad hoc* document retrieval [3]. OSIRRC 2019's vision is to build Docker-based[1] infrastructures to replicate results on standard *ad hoc* test collections. Docker is a tool that allows for the creation and deployment of applications via images containing all the required dependencies. Relying on a Docker-based infrastructure to replicate the results of existing systems, helps researchers to avoid all the issues related to system requirements and dependencies. Indeed, *Information Retrieval (IR)* platforms such as Anserini [25], Terrier [17], or text matching libraries such as MatchZoo [5] rely on a set of software tools, developed in `Java` or `Python` and based on numerous libraries for scientific computing, which have all to be available on the host machine in order for the applications to run smoothly.

Therefore, OSIRRC 2019 aims to ease the use of such platforms, and of retrieval approaches in general, by providing Docker images that replicate IR models on *ad hoc* document collections. To maximize the impact of such an effort, OSIRRC 2019 sets three main goals:

(1) **Develop** a common Docker interface specification to support images that capture systems performing *ad hoc* retrieval experiments on standard test collections. The proposed solution 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 these efforts to include additional tasks, evaluation methodologies, and benchmark initiatives.

OSIRRC 2019 gives us the opportunity to investigate on the replicability (and reproducibility), as described in the ACM guidelines discussed in [11], of *Neural IR (NeuIR)* models – on which these issues are especially relevant. Indeed, NeuIR models are highly sensitive to parameters, hyper-parameters, and pre-processing choices that hamper researchers who want to study, evaluate, and compare NeuIR models against state-of-the-art approaches. Also, these models are usually compatible only with specific versions of the libraries that they rely on (e.g., `Tensorflow`) because these frameworks are constantly updated. The use of Docker images is a possible solution to avoid these deployment issues on different machines as it already includes all the libraries required by the contained application.

For this reason, (i) we propose a Docker architecture that can be used as a framework to train, test, and evaluate NeuIR models, and is compatible with the jig introduced by OSSIRC 2019; and, (ii) we show how this architecture can be employed to build a Docker image that replicates the *Neural Vector Space Model (NVSM)* [22], a state-of-the-art unsupervised neural model for *ad hoc* retrieval. We rely on our shared `TensorFlow`[2] implementation of NVSM [18]. The model is trained, tested, and evaluated on the TREC Robust04 collection [23]. The contributions of this work are the following:

- we present a Docker architecture for NeuIR models that is compliant with the OSSIRC 2019 jig requirements. The architecture supports three functions: index, train and search, which are the same actions that are typically performed by NeuIR models;

[1] https://www.docker.com/.

[2] https://www.tensorflow.org/.

- we share two Docker images to replicate the NVSM results on the Robust04 collection: `nvsm_cpu` which relies on one or more CPUs for its computations and it is compatible with most machines, and `nvsm_gpu` which supports parallel computing using an NVIDIA *Graphics Processing Unit (GPU)*;
- we perform extensive experimental evaluations to explore replicability challenges of NeuIR models (i.e., NVSM) with Docker.

Our NVSM Docker images are part of the OSIRRC 2019 library.[3] The source code, system runs, and additional required data can be found in [10]. The findings presented in this paper contributed to the definition of the "training" hook within the jig.

The rest of the paper is organized as follows: Section 2 presents an overview of previous and related initiatives for repeatability, replicability and reproducibility. Section 3 describes the NVSM model. Section 4 presents the Docker image, whereas Section 5 describes how to interact with the provided Docker image. Section 6 presents the experimental setup and Section 7 shows the obtained results. Finally, Section 8 discusses the outcomes of our experiments and provides insights on the replicability challenges and issues of NeuIR models.

2 RELATED WORK

Repeatability, replicability, and reproducibility are fundamental aspects of computational sciences, both in supporting desirable scientific methodology as well as sustaining empirical progress. Recent ACM guidelines[4] precisely define the above concepts as follows:

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

These guidelines have been discussed and analyzed in depth in the Dagstuhl Seminar on "Reproduciblity of Data-Oriented Experiments in e-Science" held on 24-29 January 2016 [11], which focused on the core issues and approaches to reproducibility in several fields of computer science. One of the outcomes of the seminar was the *Platform, Research goal, Implementation, Method, Actor and Data (PRIMAD)* model, which tackles reproducibility from different angles.

The PRIMAD model acts as a framework to distinguish the main elements describing an experiment in computer science, as there are many different terms related to various kinds of reproducibility [4]. The main aspects of PRIMAD are the following:

- **Research Goal (R)** characterizes the purpose of a study;
- **Method (M)** is the specific approach proposed or considered by the researcher;
- **Implementation (I)** refers to the actual implementation of the method, usually in some programming language;
- **Platform (P)** describes the underlying hard- and software like the operating system and the computer used;
- **Data (D)** consists of two parts, namely the input data and the specific parameters chosen to carry out the method;
- **Actor (A)** refers to the experimenter.

Along with the main aspects of the PRIMAD model, there are two other relevant variables that need to be taken into account for reproducibility: *transparency* and *consistency*. Transparency is the ability to verify that all the necessary components of an experiment perform as they claim; consistency refers to the success or failure of a reproducibility experiment in terms of consistent outcomes.

The PRIMAD paradigm has been adopted by the IR community, where it has been adapted to the context of IR evaluation – both system-oriented and user-oriented [6]. Nevertheless, reproducibility in IR is still a critical concept, which requires infrastructures to manage experimental data [1], off-the-shelf open source IR systems [21] and reproducible baselines [2], among others.

In this context, our contribution at OSIRRC 2019 lies between replicability and reproducibility. Indeed, by relying on the NVSM implementation available at [18], we replicate the results of a reproduced version of NVSM. Therefore, we do not completely adhere to the definition of replicability provided by the ACM – as we rely on an independent implementation of NVSM rather than the one proposed in [22]. Regardless, we believe that our contribution of a Docker architecture for NeuIR models, along with the two produced NVSM Docker images, can shed some light on the replicability and reproducibility issues of NeuIR models. Besides, the off-the-shelf nature of Docker images can help future researchers to replicate/reproduce our results easily and consistently.

3 NEURAL VECTOR SPACE MODEL

The Neural Vector Space Model (NVSM) is a state-of-the-art unsupervised model for *ad hoc* retrieval. The model achieves competitive results against traditional lexical models, like *Query Language Models (QLM)* [26], and outperforms state-of-the-art unsupervised semantic retrieval models, like the Word2Vec-based models presented in [24]. Below, we describe the main characteristics of NVSM.

Given a document collection $D = \{d_j\}_{j=1}^{M}$ and the relative lexicon $V = \{w_i\}_{i=1}^{N}$, NVSM considers the vector representations $\vec{w}_i \in \mathbb{R}^{k_w}$ and $\vec{d}_j \in \mathbb{R}^{k_d}$, where k_w and k_d denote the dimensionality of word and document vectors. Due to the different size of word and document embeddings, the word feature space is mapped to the document feature space through a series of transformations learned by the model.

A sequence of n words (i.e. an n-gram) $(w_{j,i})_{i=1}^{n}$ extracted from d_j is represented by the average of the embeddings of the words in it. NVSM learns word and document representations considering mini-batches B of (n-gram, document) pairs. These representations are learned minimizing the distance between a document embedding and the representations of the n-grams contained in it. During the training phase, L2-regularization is employed to normalize the n-grams representations: $norm(\vec{x}) = \frac{\vec{x}}{||\vec{x}||}$. This process is used to obtain sparser representations. The projection of an n-gram into the k_d-dimensional document feature space can be defined as a

[3] https://github.com/osirrc/osirrc2019-library/#NVSM.
[4] https://www.acm.org/publications/policies/artifact-review-badging/.

composition function:

$$\tilde{T}((w_{j,i})_{i=1}^{n}) = (f \circ norm \circ g)((w_{j,i})_{i=1}^{n}). \quad (1)$$

Then, the standardized projection of the n-gram representation is obtained by estimating the per-feature sample mean and variance over batch B as follows:

$$T((w_{j,i})_{i=1}^{n}) = \text{hard-tanh}\left(\frac{\tilde{T}((w_{j,i})_{i=1}^{n}) - \hat{\mathbb{E}}[\tilde{T}((w_{j,i})_{i=1}^{n})]}{\sqrt{\hat{\mathbb{V}}[\tilde{T}((w_{j,i})_{i=1}^{n})]}} + \beta\right). \quad (2)$$

The composition function g, combined with the L2-normalization *norm*, causes words to compete for contributing to the resulting n-gram representation. In this way, words that are more representative for the target document will contribute more to the n-gram representation. Moreover, the standardization operation forces n-gram representations to differentiate themselves only in the dimensions that matter for the matching task. Thus, word representations incorporate a notion of term specificity during the learning process.

The similarity of two representations in the latent vector space is computed as:

$$P(\mathcal{S}|d_j, (w_{j,i})_{i=1}^{n}) = \sigma(\vec{d_j} \cdot T((w_{j,i})_{i=1}^{n})), \quad (3)$$

where $\sigma(t)$ is the sigmoid function and $\mathcal{S}$ is a binary indicator that states whether the representation of document d_j is similar to the projection of its n-gram $(w_{j,i})_{i=1}^{n}$. The probability of a document d_j given its n-gram $(w_{j,i})_{i=1}^{n}$ is then approximated by uniformly sampling z contrastive examples [13]:

$$\log \tilde{P}(d_j|(w_{j,i})_{i=1}^{n})) = \frac{z+1}{2z}\left(z \log P(\mathcal{S}|d_j, (w_{j,i})_{i=1}^{n})) + \sum_{\substack{k=1,\\ d_k \sim U(D)}}^{z} \log(1.0 - P(\mathcal{S}|d_k, (w_{j,i})_{i=1}^{n}))\right), \quad (4)$$

where $U(D)$ represents the uniform distribution used to obtain contrastive examples from documents D. Finally, to optimize the model, the following loss function – averaged over the instances in batch B – is used:

$$L(\theta|B) = -\frac{1}{m}\sum_{j=1}^{m} \log \tilde{P}(d_j|(w_{j,i})_{i=1}^{n}) + \frac{\lambda}{2m}\left(\sum_{i=1}^{|V|} ||\vec{w}_i||_2^2 + \sum_{j=1}^{|D|} ||\vec{d}_j||_2^2 + ||W||_F^2\right), \quad (5)$$

where θ is the set of parameters $\{\vec{w}_i\}_{i=1}^{|V|}$, $\{\vec{d}_j\}_{j=1}^{|D|}$, W, β and λ is a weight regularization hyper-parameter.

After training, a query q is projected into the document feature space by the composition of the functions f and g: $(f \circ g)(q) = h(q)$. Finally, the matching score between a document d_j and a query q is given by the cosine similarity of their representations in the document feature space.

4 DOCKER IMAGE ARCHITECTURE

4.1 NVSM Docker Image with CPU support

The NeuIR model we share, i.e. NVSM, is written in Python and relies on `Tensorflow v.1.13.1`. For this reason, we share a Docker image based on the official Python 3.5 runtime container, on top of which we install the Python packages required by the algorithm – such as `Tensorflow`, `Python NLTK`, and `Whoosh` – we also install a C compiler, i.e. `gcc`, in order to use the official `trec_eval` package[5] to evaluate the retrieval model during training.

Since this docker image still relies for some functions (i.e. random number generation) on the host machine, despite being very similar, the results are not exactly the same across different computers – while they are consistent on the same machine. In order to enable the replication of our experimental results, we share the model we trained with the `nvsm` image – which can be loaded in the shared Docker model if the user decides to skip the training step – in a public repository.[6]

4.2 NVSM Docker Image with GPU support

Our implementation of NVSM is based on `Tensorflow`, which is a machine learning library that allows to employ the GPU on the host machine in order to perform operations more efficiently. For this reason, we created and share in our repository another docker image (i.e. `nvsm_gpu`) which can use the GPU on the host machine to speed up the computations of the algorithm. To do so, the host machine running this image needs an NVIDIA GPU and the `nvidia-docker` version[7] installed on it. There are many advantages of employing GPUs for scientific computations, but their usage makes a sizeable difference especially when training deep learning models. The training of such models requires in fact to perform a large number of matrix operations that can be easily parallelized and do not require powerful hardware. For these reasons, the architecture of GPUs with thousands of low-power processing units is particularly indicated for this kind of operations.

The `nvsm_gpu` image is based on the official `TensorFlow-gpu` Docker image for `Python 3`. As in the other image that we share, we include a `C` compiler in order to use `trec_eval` for the retrieval model evaluation and the `Python` libraries required by NVSM.

In our experiments, we observed that `nvsm_gpu` does not produce fully consistent results on the same machine. In fact, `TensorFlow` uses the `Eigen` library, which in turn uses CUDA atomic functions to implement reduction operations, such as `tf.reduce_sum` etc. Those operations are non-deterministic and each operation can introduce small variations, as also stated in a GitHub issue on `TensorFlow` source code.[8] Despite this problem, we still believe that the advantages brought by the usage of a GPU in terms of reduction of computational time – combined with the fact that we detected only very small variations in the *Mean Average Precision at Rank 1000 (MAP)*, *Normalized Discounted Cumulative Gain at Rank 100 (nDCG@100)*, *Precision at Rank 10 (P@10)*, and *Recall* – make this implementation of the algorithm a valid alternative to the CPU-based one.

[5] https://github.com/usnistgov/trec_eval.
[6] http://osirrc.dei.unipd.it/sample_trained_models_nvsm_cpu_gpu/
[7] https://github.com/NVIDIA/nvidia-docker.
[8] https://github.com/tensorflow/tensorflow/issues/3103.

In order to assess the amount of variability due to this non-determinism in the training process we perform, in Section 7, a few tests to evaluate the difference between the run computed with `nvsm_cpu` and three different runs computed with `nvsm_gpu` using the GPU. Finally, to ensure the replicability of our experiments we share one of the models that we trained using this image on our public repository.[9] This model can be loaded by the provided Docker image to perform search on the Robust04 collection, skipping the training step.

5 INTERACTION WITH THE DOCKER IMAGE

The interaction with the shared Docker image is performed via the jig.[10] The jig is an interface to perform retrieval operations employing a model embedded in a Docker image. At the time of writing, our image can perform retrieval on the Robust04 collection. The actions supported by the NVSM Docker images that we share are: index, train, and search.

5.1 Index

The purpose of this action is to build the collection indices required by the retrieval model. Before the hook is run, the jig will mount the selected document collection at a path passed to the script. Our image will then uncompress and index the collection using the Whoosh retrieval library.[11] The index files relative to the collection are finally saved inside the Docker image in order to speed up future search operations, eliminating the time to mount the index files in the Docker image.

5.2 Train

The purpose of the train action is to train a retrieval model. We developed this hook within the jig in order to perform training with the NVSM model, which is the only NeuIR model officially supported by the OSSIRC library at the time of writing. This hook mounts the topics and relevance judgments associated to the selected experimental collection and two files containing the topic IDs to use for the test and validation of the model. The support for the evaluation of a model on different subsets of topics can be useful to any supervised NeurIR model which might use the jig in the future or to learning-to-rank approaches within other Docker images. In the case of NVSM, which is an unsupervised model, we employ the validation subset of topics during training to select the best model – saved after each training epoch. The trained models are saved to a directory indicated by the user on the host machine. This is done in order to keep the Docker image as light as possible, and to allow the user to easily inspect the results of the training process.

5.3 Search

The purpose of the search hook is to perform an ad-hoc retrieval run – multiple runs can be performed by calling jig multiple times with different parameters. In order to perform retrieval with the provided Docker image, the user will need to indicate as a parameter the path to the directory containing the trained model computed at the previous step and the path to a text file containing the topic IDs on which to perform retrieval. Then, the NVSM trained model – which performed best on the validation set of queries at the previous step – will be loaded by the Docker image and retrieval will be performed on the topics specified in the topic IDs file passed to the jig.

6 EXPERIMENTAL SETUP

6.1 Experimental Collection

To test our docker image we consider the Robust04 collection, which is composed of TIPSTER corpus [14] Disk 4&5 minus CR. The collection counts 528,155 documents, with a vocabulary of 760,467 different words. The topics considered for the evaluation are topics 301-450, 601-700 from Robust04 [23]. Only the field *title* of topics is used for retrieval. The set of topics is split into validation (V) and test (T) sets, as proposed in [22].[12] Relevance judgments are restricted accordingly.

The execution times and memory occupation statistics were computed on an 2018 Alienware Area-51 with an Intel Core i9-7980XE CPU @ 2.60GHz with 36 cores, 64GB of RAM and two GeForce GTX 1080Ti GPUs.

6.2 Evaluation Measures

We use the same measures of [22] to evaluate retrieval effectiveness: MAP, nDCG@100, P@10. Additionally, we also employ Recall.

6.3 Training

To train the NVSM model, we set the following parameters and hyper-parameters: word representation size $k_w = 300$, number of negative examples $z = 10$, learning rate $\alpha = 0.001$, regularization lambda $\lambda = 0.01$, batch size $m = 51200$, dimensionality of the document representations $k_d = 256$ and n-gram size $n = 16$. We train the model for 15 iterations over the document collection and we select the model iteration that performs best in terms of MAP. A single iteration consists of $\lceil \frac{1}{m} \sum_{d \in D} (|d| - n + 1) \rceil$ batches, where d is a document in the experimental collection D, and n is the n-gram size as described in Section 3.

6.4 Performance Differences Evaluation

In order to evaluate the differences between the rankings produced with the two NVSM Docker images we consider the following measures.

- **Root Mean Square Error (RMSE)**: This measure indicates how close are the performance scores of two systems [16] (the lower the better) and considers the values associated to a measure $M(\cdot)$ (i.e. MAP, nDCG@100) chosen for their evaluation. RMSE is defined as follows:

$$RMSE = \sqrt{\frac{1}{T} \sum_{i=1}^{T} (M_{0,i} - M_{1,i})^2}, \tag{6}$$

  where $M_{0,i}$ is the chosen evaluation measure associated to the first run to compare and $M_{1,i}$ is the same measure associated to the second run, both relative to the i^{th} topic.

[9] http://osirrc.dei.unipd.it/sample_trained_models_nvsm_cpu_gpu/.
[10] https://github.com/osirrc/jig.
[11] https://whoosh.readthedocs.io/en/latest/index.html.
[12] Splits can be found at: https://github.com/osirrc/jig/tree/master/sample_training_validation_query_ids.

- **Kendall's τ correlation coefficient**: since different runs may produce the same RMSE score, we also measure how close are the ranked results lists of two systems. This is measured using Kendall's τ correlation coefficient [15] among the list of retrieved documents for each topic, averaged across all topics. The Kendall's τ correlation coefficient on a single topic is given by:

$$\tau_i(run_0, run_1) = \frac{P - Q}{\sqrt{(P + Q + U)(P + Q + V)}}, \quad (7)$$

where T is the total number of topics, P is the total number of concordant pairs (document pairs that are ranked in the same order in both vectors) Q the total number of discordant pairs (document pairs that are ranked in opposite order in the two vectors), U and V are the number of ties, respectively, in the first and in the second ranking. To compare two runs, Equation (7) becomes:

$$\overline{\tau}(run_0, run_1) = \frac{1}{T} \sum_{i=1}^{T} \tau_i(run_0, run_1) \quad (8)$$

The range of this measure is [-1, 1], where 1 indicates a perfect correlation between the order of the documents in the considered runs and -1 indicates that the rankings associated to each topic in the runs are one the inverse of the other.

- **Jaccard index**: since different runs might contain a different set of relevant documents for each topic, we consider the average of the Jaccard index of each of these sets, over all topics. We compute this value as:

$$sim(run_1, run_2) = \frac{1}{T} \sum_{i=1}^{T} \frac{|rd\_run1_i \cap rd\_run2_i|}{|rd\_run1_i \cup rd\_run2_i|} \quad (9)$$

where rd_run1_i and rd_run2_i are the sets of relevant documents retrieved for the topic i in $run1$ and $run2$, respectively.

RMSE and Kendall's τ correlation coefficient have been adopted to evaluate the differences between the rankings for reproducibility purposes in CENTRE@CLEF [9], whereas the Jaccard index is used here for the first time for this purpose.

We use these measures to evaluate the differences in the rankings produced by the NVSM Docker images. Since the image with GPU support is not fully deterministic we compute three different runs on the same machine and analyze the differences between them. Also, since NVSM does not retrieve any relevant document for four topics (312, 316, 348 and 379) because none of their terms are present in the NVSM term dictionary, we remove these topics from the pool and consider – only for the comparison between different runs using RMSE, Kendall's τ correlation coefficient and Jaccard index – a total of 196 out of the 200 test topics indicated in the test split file available in the OSSIRC repository.[13]

7 EVALUATION

In Table 1, we report the statistics relative to the disk space and memory occupation of our images. We also include the time required by each image to complete one training epoch. The first thing that we observe is that the CPU Docker image takes less space on disk than the GPU one. This is because the former does not need all of the drivers and libraries required by the GPU version of `Tensorflow`. In fact, these libraries make the `nvsm_gpu` image three times larger than the other one. We also point out that the GPU memory usage reported in Table 1 is proportional to the memory available on the GPU in the host machine. In fact, `Tensorflow` – if there is no explicit limitation imposed by the user – allocates most of the available space in the GPU memory to speed up computations. For this reason, since the GPU used in these experiments has 11GB of memory available, the space used is 10.76GB. If the GPU had less space available, then `Tensorflow` would be able to adjust to it and use as low as 8GB. This is in fact the minimum GPU memory requirement according to our tests, to run the `nvsm_gpu` Docker image.

	NVSM (CPU)	NVSM (GPU)
Disk occupation (image only)	1.1GB	3.55GB
Index disk size (Robust04)	4.96GB	
Maximum RAM occupation	16GB	10GB
GPU memory usage	–	10.76GB
Execution time (1 epoch)	8h	2h30m

Table 1: Analysis of the space on disk used, memory occupation and execution time of the shared docker images.

In Table 2, we report the retrieval results obtained with the two shared Docker images. From these results, we observe that there are small differences, always within ±0.01, between the runs obtained with `nvsm_gpu` on the same machine and with the ones obtained with `nvsm_cpu` on different machines. The causes of these small differences are described in Section 4 and are related to how the optimization process of the model is managed by `Tensorflow` within Docker.

	MAP	nDCG@100	P@10	Recall
CPU (run 0)	0.138	0.271	0.285	0.6082
GPU (run 0)	0.137	0.265	0.277	0.6102
GPU (run 1)	0.138	0.270	0.277	0.6066
GPU (run 2)	0.137	0.268	0.270	0.6109

Table 2: Retrieval results on the Robust04 (T) collection computed with the two shared Docker images of NVSM.

The MAP, nDCG@100, P@10, and Recall values obtained with the images are all very similar, and close to the measures reported in the original NVSM paper [22]. Indeed, the absolute difference between the reported MAP, nDCG@100, and P@10 values in [22] and our results is always less than 0.02. As a side note, the MAP values obtained by NVSM are low when compared to the other approaches on Robust04 that can be found in the OSIRRC 2019 library – even 10% lower than some methods that do not apply re-ranking.

In order to further evaluate the performance differences between the runs, we begin computing the RMSE considering the MAP, nDCG@100, and P@10 measures. The RMSE gives us an idea of the performance difference between two runs – averaged across

[13] https://github.com/osirrc/jig/tree/master/sample_training_validation_query_ids.

the considered topics. We first compute the average values of MAP, nDCG@100, and P@10 over the three nvsm_gpu runs on each topic. Then, we compare these averaged performance measures, for each topic, against the corresponding ones associated to the CPU-based NVSM run we obtained on our machine. These results are reported in Table 3. From the results of this evaluation we can observe that

	NVSM GPU (average)
RMSE (MAP)	0.034
RMSE (nDCG@100)	0.054
RMSE (P@10)	0.140

Table 3: RMSE (the lower the better) between the NVSM CPU Docker image and the average of the 3 runs computed with the NVSM GPU Docker image considering the MAP at rank 1000 (MAP), nDCG at rank 100 (nDCG@100), and Precision at rank 10 (P@10) between the NVSM CPU run and the average of the three runs computed with the NVSM Docker image supporting GPU computations.

the average performance difference across the considered 196 topics is very low when considering the MAP and nDCG@100 measures, while it grows when we consider the top part of the rankings (P@10). In conclusion, the RMSE value is generally low, hence we can confidently say that the models behave in a very similar way in terms of MAP, nDCG@100, and P@10 on all the considered topics.

In Table 4, we report the Kendall's τ measures associated to each pair of runs that we computed. This measure shows us how much the considered rankings are similar to each other. In our case, the runs appear to be quite different from each other, since the Kendall's τ values are all close to 0. In other words, when considering the top 100 results in each run, the same documents are rarely in the same positions in the selected rankings. This result, combined with the fact that the runs achieve all similar MAP, nDCG@100, P@10, and Recall values, leads to the conclusion that the relevant documents are ranked high in the rankings, but are not in the same positions. In other words, NVSM performs a permutation of the documents in the runs, maintaining however the relative order between relevant and non-relevant documents.

	GPU (run 0)	GPU (run 1)	GPU (run 2)	CPU
GPU (run 0)	1.0	0.025	0.025	0.018
GPU (run 1)	0.025	1.0	0.089	0.014
GPU (run 2)	0.025	0.089	1.0	0.009
CPU	0.018	0.014	0.009	1.0

Table 4: Kendall's τ correlation coefficient values between the runs we computed with the NVSM GPU and CPU Docker images considering the top 100 ranked documents in each run.

To validate our hypothesis, we report in Figure 1, for each pair of runs, the Jaccard index between the sets of relevant documents averaged over all topics, as described in Section 6. These values help us to assess whether the set of relevant documents retrieved for each topic in our runs is different, and by how much on average. In this case, we observe that the runs computed with the GPU have more in common between each other than the run computed with the CPU. However, the Jaccard index values are all very high and this confirms our previous hypothesis about the rankings. In fact, this implies that they contain similar sets of relevant documents, which are however in different relative positions – because we have a low Kendall's τ correlation coefficient – but in the same portion of the rankings – because we obtain similar and relatively high nDCG@100, P@10 values over all runs.

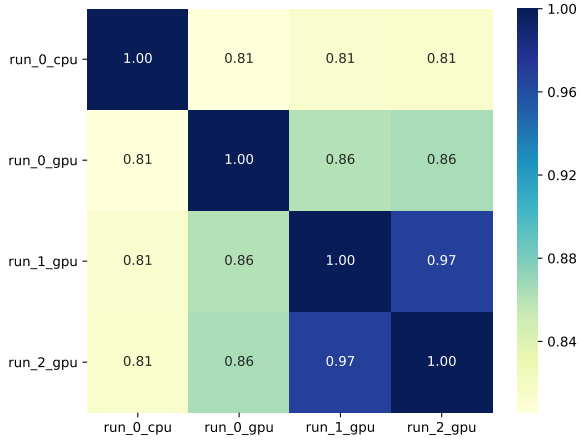

Figure 1: Heatmap of the average Jaccard index between the sets of retrieved relevant documents for each topic by the NVSM Docker images.

To qualitatively assess from a user perspective the differences between the different runs we select one topic (301: "International Organized Crime") and report in Table 5 the top five document ids for each of them. The results in this table confirm our previous

CPU	GPU (run 0)	GPU (run 1)	GPU (run 2)
FBIS3-55219	**FBIS3-55219**	**FBIS3-55219**	**FBIS3-55219**
FBIS4-41991	**FBIS4-7811**	**FBIS4-7811**	**FBIS4-7811**
FBIS4-45469	**FBIS4-43965**	FBIS4-41991	FBIS4-41991
FBIS3-54945	**FBIS3-23986**	**FBIS3-23986**	**FBIS3-23986**
FBIS4-7811	FBIS4-41991	FBIS4-65446	FBIS4-65446

Table 5: Top 5 documents in the runs computed with nvsm_cpu and nvsm_gpu. Relevant documents are highlighted in bold.

intuition. In fact, we observe that the majority of the high-ranked documents for topic 301 for each run are relevant, but these documents are slightly different across different runs. Also, we observe that the most of the relevant documents retrieved by nvsm_gpu are the same, while only two of the relevant documents retrieved by nvsm_cpu are also found in the other runs. For instance, we observe that document FBIS4-45469 is ranked in the top-5 positions only in the CPU run. Similarly, document FBIS4-43965 appears only in the GPU run 0. These apparently small differences can have a sizeable impact on the user experience and should be taken into

consideration when choosing to employ a NeuIR model in real-case scenarios.

8 FINAL REMARKS

In this work, we performed a replicability study of the *Neural Vector Space Model (NVSM)* retrieval model using Docker. First, we presented the architecture and the main functions of a Docker image designed for the replicability of *Neural IR (NeuIR)* models. The described architecture is compatible with the *jig* developed in the OSSIRC 2019 workshop and supports the index (to index an experimental collection), train (to train a retrieval model), and search (to perform document retrieval) actions. Secondly, we described the image components and the engineering challenges to obtain deterministic results with Docker using popular machine learning libraries such as `Tensorflow`. We also share two Docker images – which are part of the OSSIRC 2019 library – of the NVSM model: the first, which relies only on the CPU of the host machine to perform its operations, the second, which is able to also exploit the GPU of the host machine, when available, to perform more expensive computations such as the training of the NVSM model. Finally, we performed an in-depth evaluation of the differences between the runs obtained with the two images, presenting some insights which also hold for other NeuIR models relying on CUDA and `Tensorflow`.

In fact, we observed some differences – which are hard to spot when looking only at the average performance – between the runs computed by the `nvsm_cpu` Docker images on different machines and between the runs computed by the `nvsm_cpu` and `nvsm_gpu` Docker images on the same machine. The differences between `nvsm_cpu` images on different machines are related to the non-determinism of the results, as Docker relies on the host machine for some basic operations which influence the model optimization process through the generation of different pseudo-random number sequences. On the other hand, the differences between `nvsm_gpu` images on the same machine are due to the implementation of some functions in the CUDA and `Tensorflow` libraries. We observed that these operations influence in a sizeable way the ordering of the same documents across different runs, but not the overall distribution of relevant and non-relevant documents in the ranking. Similar differences, that are even more accentuated, can be found between `nvsm_cpu` and `nvsm_gpu` images on the same machine. Therefore, even though these differences may seem marginal in offline evaluation settings, where the focus is on average performance, they are extremely relevant for user-oriented online settings – as they can have a sizeable impact on the user experience and should thus be taken into consideration when deciding whether to use NeuIR models in real-world scenarios.

We also share the models we trained on our machine with both the `nvsm_cpu` and `nvsm_gpu` Docker images in our public repository, as it is fundamental to enable replicability [12]. These can be loaded by the docker image in order to perform document retrieval with the same models we used and obtain the same runs.

REFERENCES

[1] M. Agosti, N. Ferro, and C. Thanos. 2012. DESIRE 2011: workshop on data infrastructurEs for supporting information retrieval evaluation.. In *SIGIR Forum*, Vol. 46. Citeseer, 51–55.

[2] J. Arguello, M. Crane, F. Diaz, J. Lin, and A. Trotman. 2016. Report on the SIGIR 2015 workshop on reproducibility, inexplicability, and generalizability of results (RIGOR). In *ACM SIGIR Forum*, Vol. 49. ACM, 107–116.

[3] R. Clancy, N. Ferro, C. Hauff, J. Lin, T. Sakai, and Z. Z. Wu. 2019. The SIGIR 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). (2019).

[4] D. De Roure. 2014. The future of scholarly communications. *Insights* 27, 3 (2014).

[5] Y. Fan, L. Pang, J. Hou, J. Guo, Y. Lan, and X. Cheng. 2017. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270* (2017).

[6] N. Ferro, N. Fuhr, K. Järvelin, N. Kando, M. Lippold, and J. Zobel. 2016. Increasing Reproducibility in IR: Findings from the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science". *SIGIR Forum* 50, 1 (June 2016), 68–82. https://doi.org/10.1145/2964797.2964808

[7] N. Ferro, N. Fuhr, M. Maistro, T. Sakai, and I. 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)*.

[8] N. Ferro and D. Kelly. 2018. SIGIR Initiative to Implement ACM Artifact Review and Badging. *SIGIR Forum* 52, 1 (Aug. 2018), 4–10. https://doi.org/10.1145/3274784.3274786

[9] N. Ferro, M. Maistro, T. Sakai, and I. Soboroff. 2018. Overview of CENTRE@CLEF 2018: A first tale in the systematic reproducibility realm. In *International Conference of the Cross-Language Evaluation Forum for European Languages*. Springer, 239–246.

[10] N. Ferro, S. Marchesin, A. Purpura, and G. Silvello. 2019. Docker Images of Neural Vector Space Model. https://doi.org/10.5281/zenodo.3246361

[11] J. Freire, N. Fuhr, and A. Rauber. 2016. Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041). *Dagstuhl Reports* 6, 1 (2016), 108–159. https://doi.org/10.4230/DagRep.6.1.108

[12] N. Fuhr. 2018. Some common mistakes in IR evaluation, and how they can be avoided. In *ACM SIGIR Forum*, Vol. 51. ACM, 32–41.

[13] M. Gutmann and A. Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. of the 13th International Conference on Artificial Intelligence and Statistics*. 297–304.

[14] D. Harman. 1992. The DARPA tipster project. In *ACM SIGIR Forum*, Vol. 26. ACM, 26–28.

[15] M. G. Kendall. 1948. Rank correlation methods. (1948).

[16] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. 2016. Toward reproducible baselines: The open-source IR reproducibility challenge. In *European Conference on Information Retrieval*. Springer, 408–420.

[17] C. Macdonald, R. McCreadie, R. L. Santos, and I. Ounis. 2012. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR* (2012), 60–63.

[18] S. Marchesin, A. Purpura, and G. Silvello. 2019. A Neural Vector Space Model Implementation Repository. https://github.com/giansilv/NeuralIR/

[19] T. Sakai, N. Ferro, I. Soboroff, Z. Zeng, P. Xiao, and M. Maistro. 2019. Overview of the NTCIR-14 CENTRE Task. In *Proceedings of the 14th NTCIR Conference on Evaluation of Information Access Technologies. Tokyo, Japan.*

[20] I. Soboroff, N. Ferro, and T. Sakai. 2018. Overview of the TREC 2018 CENTRE Track. In *The Twenty-Seventh Text REtrieval Conference Proceedings (TREC 2018)*.

[21] A. Trotman, C. L. A. Clarke, I. Ounis, S. Culpepper, M. A. Cartright, and S. Geva. 2012. Open source information retrieval: a report on the SIGIR 2012 workshop. In *ACM SIGIR Forum*, Vol. 46. ACM, 95–101.

[22] C. Van Gysel, M. de Rijke, and E. Kanoulas. 2018. Neural Vector Spaces for Unsupervised Information Retrieval. *ACM Trans. Inf. Syst.* 36, 4 (2018), 38:1–38:25.

[23] E. M. Voorhees. 2005. The TREC Robust Retrieval Track. *ACM SIGIR Forum* 39, 1 (2005), 11–20.

[24] I. Vulić and M. F. Moens. 2015. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. ACM, 363–372.

[25] P. Yang, H. Fang, and J. Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *J. Data and Information Quality* 10, 4, Article 16 (Oct. 2018), 20 pages. https://doi.org/10.1145/3239571

[26] C. Zhai and J. Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)* 22, 2 (2004), 179–214.