Continuous Integration for Reproducible Shared Tasks with TIRA.io

Maik Fröbe[1(✉)], Matti Wiegmann[2], Nikolay Kolyada[2], Bastian Grahm[3], Theresa Elstner[3], Frank Loebe[3,4], Matthias Hagen[1], Benno Stein[2], and Martin Potthast[3,4]

[1] Friedrich-Schiller-Universität Jena, Jena, Germany
maik.froebe@uni.jena.de
[2] Bauhaus-Universität Weimar, Weimar, Germany
[3] Leipzig University, Leipzig, Germany
[4] ScaDS.AI, Leipzig, Germany

Abstract. A major obstacle to the long-term impact of most shared tasks is their lack of reproducibility. Often only the test collections and the papers of the organizers and participants are published. Third parties who want to independently evaluate the state of the art for a task on other data must re-implement the participants' software. The tools developed to collect software from participants in shared tasks only partially verify its reliability at the time of submission, much less long-term, and do not enable third parties to reuse it later. We have overhauled the TIRA Integrated Research Architecture to address all of these issues. The new version simplifies task setup for organizers and software submission for participants, scales from a local computer to the cloud, supports on-demand resource allocation up to parallel CPU and GPU processing, and enables export for local reproduction with just a few lines of code. This is achieved by implementing the TIRA protocol with an industry-standard continuous integration and deployment (CI/CD) pipeline using Git, Docker, and Kubernetes.

1 Introduction

A shared task is a collaborative laboratory experiment to evaluate state-of-the-art computational solutions to a problem, the task. A reproducible shared task gathers the resources needed by third parties to reproduce the evaluation results. Reproducibility is only guaranteed if the datasets of the shared tasks and the software of the individual participants are available. However, most participants in shared tasks do not publish their software, and most organizers do not have the time or resources to collect it. Therefore, the results of shared tasks are usually difficult for third parties to reproduce after a shared task has been completed.

J. Kamps et al. (Eds.): ECIR 2023, LNCS 13982, pp. 236–241, 2023.
https://doi.org/10.1007/978-3-031-28241-6_20

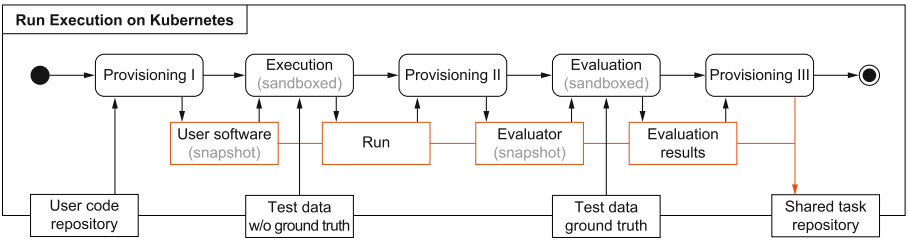

Fig. 1. UML activity diagram of the control flow (rounded boxes) and the data flow (rectangular boxes). All run artifacts (highlighted middle "row") are persisted through the provisioning steps in the shared task repository. The user software and evaluator are "untrusted" software and therefore run in a sandbox without network access.

In information retrieval, many initiatives have been launched to promote the reproducibility of experiments and software. These include, among others, the reproducibility and resource tracks at the IR conferences, the OSIRRC workshop [2], as well as the CENTRE lab held in close succession at CLEF [4], TREC [13], and NTCIR [12]. Moreover, the ACM SIGIR Artifact Badging Board [3] awards badges of honor to especially reproducible papers. With respect to reproducible experiments and shared tasks, a requirements catalog has been developed at the Evaluation-as-a-Service (EaaS) workshop [7], drawing inspiration from existing tools [5,6,14,15], as a guide to future ones [1,8,10,11,16].

In this paper, we present a new approach to make shared tasks reproducible based on continuous integration and deployment (CI/CD), a widely used industry standard in software engineering. In the context of research software engineering for shared tasks, CI refers to the reproducible evaluation of each software version (cf. Sect. 2), while CD refers to the automatic archiving of each evaluated software version in a centralized shared task repository (cf. Sect. 3). Our new approach is based on modern, widely used industry-standard tools, including GitHub and/or GitLab, Docker, and the cluster computing framework Kubernetes. This minimizes the effort for organizers and participants alike and allows for instant reproduction, all with just a few lines of code.

2 Snapshotting Software on Every Run

Reproducibility is always a trade-off between cost and benefit: Perfect reproducibility may, in extreme cases, require persistence of the hardware on which a piece of software was executed. However, implementing a practical form of reproducibility is already quite a challenge. Originally, TIRA persisted software through virtual machine snapshots [5], which was prone to scaling errors due to the VirtualBox implementation, sometimes requiring costly manual intervention. Recent progress in DevOps and continuous integration and deployment, well integrated with Git, provide all the components for automating reproducible shared tasks with mature and cloud-native tools. Consequently, our new backend

for TIRA is cloud-native and based on a privately hosted GitLab, its container registry (12.4 PB HDD storage), and a Kubernetes cluster (1,620 CPU cores, 25.4 TB RAM, 24 GeForce GTX 1080 GPUs).

Figure 1 provides an overview of TIRA's new workflow for shared tasks. The workflow is based on a Git repository for shared tasks and uses the tools built into Git platforms: (1) a user code repository with a dedicated container registry where participants upload Docker images, (2) continuous integration runners that execute the five phases of the pipeline, (3) Kubernetes to orchestrate (provision, scale, and distribute) the runners in the cluster, and (4) a storage platform that provides access to the datasets.

First, users upload their software (in a Docker image) to the container registry in their private code repository, which is maintained on the Git platform. Access is granted at the time of login to TIRA via an automatically generated authentication token. Then, users can add their software to TIRA by specifying the command to execute the software in the Docker image. Any software added in this way is immutable: the command and Docker image cannot be changed afterwards (participants can upload as many pieces of software as they like). The container registry has a lower memory footprint compared to virtual machines. It is also private during the task (only the user and TIRA have access) to avoid premature publication [9].

The workflow for executing and evaluating software is specified via the declarative continuous integration API in the shared task's Git repository, which also contains a registry for the evaluator images. The workflow is triggered by a special commit in the shared task repository (specifying the software to be executed; done via the TIRA website or the command line) and consists of five steps:

1. *Provisioning I*: Prepares the execution environment by branching and cloning the shared task repository and copying the test data to the execution environment; all operations are trusted.
2. *Execution*: In the prepared execution environment with the test data, this step moves the user software into a sandbox and then executes it to generate its output as a so-called run file. Sandboxing cuts off the internet connection (using egress and ingress rules in Kubernetes) to ensure that (untrusted) user software does not leak test data. This enforces blind evaluation and ensures that the software is mature enough to run unattended in its Docker image (i.e., the software cannot download any data while it is running).
3. *Provisioning II*: Persists the run files and logs, and copies the test ground truth to the execution environment for evaluation; all operations are trusted.
4. *Evaluation*: In the prepared execution environment with the run files and the test truth, the evaluator is executed in a sandbox to generate the evaluation results. The evaluator is not trusted since it is an external software.
5. *Provisioning III*: Persists the evaluation results and logs and merges the executed software's branch into the main branch (all operations are trusted).

This workflow ensures that only the data of successfully executed and evaluated software is in the main branch of the Git repository of the shared task. Branches indicate queued or running software. Since the software itself is immutable, every software is snapshotted on every run.

```
import tira
df = tira.load_data('<dataset-name>')
# df can be manipulated for ablation/replicability/reproducibility studies
predictions, evaluation = tira.run(
    '<task-name>/<user-name>/<software-name>',
    data=df, evaluate='<evaluator-name>'
)
```

Listing 1: The software `<software-name>` by user `<user-name>` submitted in the `<task-name>` shared task is executed and evaluated on a pandas DataFrame `df` of dataset `<dataset-name>`. Demo available at tira.io/t/post-hoc-experimentation.

3 Repeat, Replicate, and Reproduce in One Line of Code

Through continuous deployment (CD), our new version of TIRA provides organizers with a self-contained Git repository that contains all shared tasks artifacts and is ready to be published. It contains all datasets, runs, evaluation results, logs, metadata and software snapshots. This "shared task repository" also contains utility scripts that allow all software submitted to the shared task to be run on the existing datasets, but also on other datasets as long as their formatting is the same. Listing 1 illustrates this by loading a (new or additional) dataset into a Pandas DataFrame, which then serves as input to software submitted to the shared task while the run output is evaluated directly. The utility script `tira` that enables this replication is in the repository; Docker and Python 3 are the only external dependencies. When archiving the shared task, all pieces of software are published to Docker Hub so that they can be loaded ad-hoc during replications (TIRA also maintains a local archive).

The final shared task repository can be easily published and serves as a natural entry point for a variety of follow-up studies. All researchers can fork such a repository and make contributions that can increase the impact of a shared task through additional material (e.g., additional datasets, evaluations, ablations, software, etc.). None of this was previously possible with the old version of TIRA or any other related tool to support reproducible experiments.

4 Conclusion

Our new version of TIRA enables reproducible shared tasks with software submissions in a cloud-native environment and is currently being used in two shared tasks at SemEval 2023. Cloud-native orchestration reduces the burden of organizing shared tasks with software submissions. Therefore, we plan to spread TIRA further, to collect more shared tasks on the platform for which post-hoc experiments are then possible, and to further encourage the submission of software in shared tasks.

In the future, we will port our pipeline to support more and also proprietary vendors (GitHub, AWS/Azure), which will make TIRA more accessible. In addition, we aim for one-click deployments that use private repositories in GitHub or (self-hosted instances of) GitLab as the backend for shared tasks.

Acknowledgments. This work has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070014 (OpenWebSearch.EU, https://doi.org/10.3030/101070014).

References

1. Breuer, T., Schaer, P., Tavakolpoursaleh, N., Schaible, J., Wolff, B., Müller, B.: STELLA: Towards a Framework for the Reproducibility of Online Search Experiments. In: Proceedings of the Open-Source IR Replicability Challenge OSIRRC@SIGIR 2019, pp. 8–11 (2019)
2. Clancy, R., Ferro, N., Hauff, C., Lin, J., Sakai, T., Wu, Z.Z.: The SIGIR 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In: Proceedings of SIGIR 2019, pp. 1432–1434 (2019)
3. Ferro, N., Kelly, D.: SIGIR initiative to implement ACM artifact review and badging. SIGIR Forum **52**(1), 4–10 (2018)
4. Ferro, N., Maistro, M., Sakai, T., Soboroff, I.: Overview of CENTRE@CLEF 2018: A First Tale in the Systematic Reproducibility Realm. In: CLEF, pp. 239–246 (2018)
5. Gollub, T., Potthast, M., Beyer, A., Busse, M., Rangel, F., Rosso, P., Stamatatos, E., Stein, B.: Recent Trends in Digital Text Forensics and its Evaluation. In: Proceedings of CLEF 2013, pp. 282–302 (2013)
6. Hopfgartner, F., et al.: Benchmarking news recommendations: the CLEF NewsREEL use case. SIGIR Forum **49**(2), 129–136 (2015)
7. Hopfgartner, F., et al.: Evaluation-as-a-Service for the Computational Sciences: Overview and Outlook. ACM J. Data Inf. Qual. **10**(4), 15:1–15:32 (2018)
8. Jagerman, R., Balog, K., de Rijke, M.: OpenSearch: Lessons Learned from an Online Evaluation Campaign. ACM J. Data Inf. Qual. **10**(3), 13:1–13:15 (2018)
9. Lin, J., Campos, D., Craswell, N., Mitra, B., Yilmaz, E.: Fostering Coopetition While Plugging Leaks: The Design and Implementation of the MS MARCO Leaderboards. In: Proceedings of SIGIR 2022, pp. 2939–2948 (2022)
10. Pavao, A.: CodaLab Competitions: An Open Source Platform to Organize Scientific Challenges. Université Paris-Saclay, France, Tech. rep. (2022)
11. Potthast, M., Gollub, T., Wiegmann, M., Stein, B.: TIRA integrated research architecture. In: Information Retrieval Evaluation in a Changing World. TIRS, vol. 41, pp. 123–160. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22948-1_5
12. Sakai, T., Ferro, N., Soboroff, I., Zeng, Z., Xiao, P., Maistro, M.: Overview of the NTCIR-14 CENTRE Task. In: Proceedings of NTCIR-14 (2019)
13. Soboroff, I., Ferro, N., Sakai, T.: Overview of the TREC 2018 CENTRE Track. In: Proceedings of TREC 2018 (2018)
14. Tsatsaronis, G., et al.: An Overview of the BIOASQ Large-scale Biomedical Semantic Indexing and Question Answering Competition. BMC Bioinform. **16**, 138:1–138:28 (2015)

15. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. SIGKDD Explor. **15**(2), 49–60 (2013)
16. Yadav, D., et al.: EvalAI: Towards Better Evaluation Systems for AI Agents. arXiv 1902.03570 (2019)