

TIRA Integrated Research Architecture



Martin Potthast, Tim Gollub, Matti Wiegmann, and Benno Stein

Abstract Data and software are immaterial. Scientists in computer science hence have the unique chance to let other scientists easily reproduce their findings. Similarly, and with the same ease, the organization of shared tasks, i.e., the collaborative search for new algorithms given a predefined problem, is possible. Experience shows that the potential of reproducibility is hardly tapped in either case. Based on this observation, and driven by the ambitious goal to find the best solutions for certain problems in our research field, we have been developing the TIRA Integrated Research Architecture. Within TIRA, the reproducibility requirement got top priority right from the start. This chapter introduces the platform, its design requirements, its workflows from both the participants' and the organizers' perspectives, alongside a report on user experience and usage scenarios.

1 Introduction

Computer science, when focusing on its data- and software-driven branches, is probably the only scientific discipline where the subject of research (the data) and its result (the software) can be copied, bundled, and shipped at virtually no extra cost—beyond what it took to acquire and create them respectively. The reproducibility of a computer science paper is greatly improved if the data and software underlying

M. Potthast (✉)
Leipzig University, Leipzig, Germany
e-mail: martin.pothast@uni-leipzig.de

T. Gollub · B. Stein
Bauhaus-Universität Weimar, Weimar, Germany
e-mail: tim.gollub@uni-weimar.de; benno.stein@uni-weimar.de

M. Wiegmann
Bauhaus-Universität Weimar, Weimar, Germany
German Aerospace Center (DLR), Jena, Germany
e-mail: matti.wiegmann@uni-weimar.de; matti.wiegmann@dlr.de

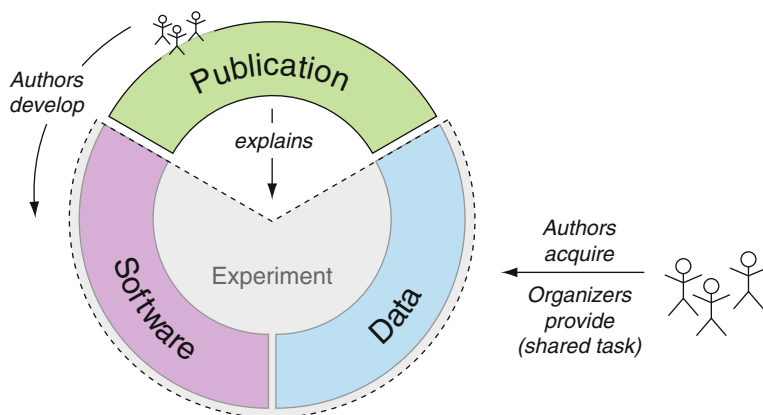


Fig. 1 The three elements of reproducibility in computer science: publication, data, software. The two latter form the experiment. The software is developed by the authors of the publication, while the data may also be provided by a third party such as the organizers of a shared task

its experiments are available for the community. The publication (the paper) closes the circle by supplying motivation for the tackled problem, high-level descriptions of the courses of action taken, interpretation of the results obtained, and theories derived from observations made; see Fig. 1 for an illustration. However, the current practice in computer science differs: data and software of an experimental setting are typically not published, although they are the only tangible evidence of the claims made in a paper. From the outside it may look odd that scientists do not expose this evidence to third-party verification, thereby effectively reducing their papers to the level of anecdotes, until someone else comes along and double-checks.

It should be noted that the current economies of science do not enforce the reproducibility ideal (Stodden 2010); the commonly accepted measures for scientific excellence do not count movable assets other than papers. Any extra effort spent on data and software, despite increasing a publication’s impact and furthering one’s reputation, does not yield sufficient returns compared to moving on to the next topic or task. Not sharing data and software, however, poses an impediment to scientific progress: when someone decides to work on a task which has been tackled at least once before—effectively rendering it a shared task—they must take into account the workload required to reproduce missing assets to compare their own approach with those proposed earlier. The effort spent here is testimony to a scientist’s diligence, but many use only a small number of approaches for comparison, and sometimes none at all. Moreover, it can be difficult to avoid biasing experimental results while juggling the conflict of interest between optimizing one’s own approach versus those of third parties. Yet, even organized shared task events currently provide only part of the solution: while benchmark datasets are shared, software is typically not.

The chapter in hand introduces the TIRA Integrated Research Architecture, TIRA, a modularized platform for shared tasks. Section 2 provides background on computer science reproducibility and outlines our understanding of how shared

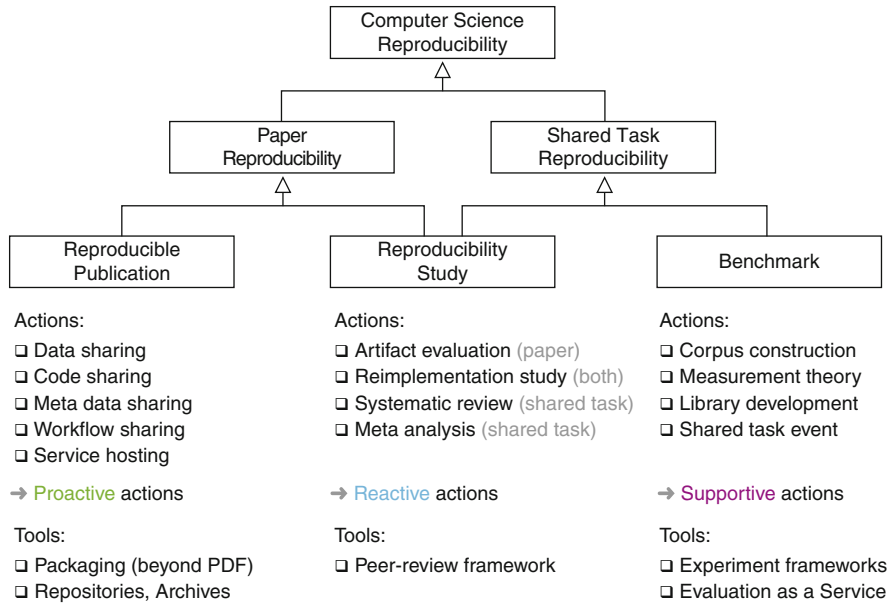


Fig. 2 Taxonomy of actions towards improving reproducibility in computer science

tasks emerge in computer science from which we derive requirements of shared task platforms. Section 3 introduces TIRA from the perspectives of participants and organizers, Sect. 4 generalizes the shared task paradigm from supporting software submission to data submission, and Sect. 5 reports how users have experienced the prototype.

2 Untangling the Reproducibility Thicket

Many initiatives across computer science seek to improve the situation with both organizational and technological means, aiming at easier sharing, citability of data and software, and better reproducibility. So far we have collected about 90 initiatives most of which are still active. Since a review of all of these initiatives and the related tools by far exceeds the scope and space limitations of this chapter, we have organized them into a taxonomy; see Fig. 2. Our high-level overview is oriented at the actions they have taken—which, by extension, every scientist can take for himself, too—to improve computer science reproducibility. TIRA is but one of these initiatives, providing a platform for shared task reproducibility. Otherwise, comprehensive overviews of relevant subsets of the existing initiatives have been recently compiled by Hanbury et al. (2015) and Freire et al. (2016).

2.1 Computer Science Reproducibility

Although reproducibility is one of the cornerstones of empirical science, its history in the empirical branches of computer science is comparably short. Claerbout and Karrenbach (1992), upon witnessing reproducibility problems within computational parts of their geophysics project, were among the first to force the issue by implementing rigorous guidelines for the project members, and publishing about them. Some followed their example and developed their own strategies (e.g., Donoho et al. 2009). However, the majority of computer scientists continued with business as usual, occasionally interrupted by some who denounced the lack of reproducibility when failing to verify published results by reimplementing (e.g., Pedersen 2008; Fokkens et al. 2013). Pioneered by Stodden et al., systematic research into computer science reproducibility started only recently,¹ in the wake of what has become known as science's reproducibility crisis,² which struck home particularly hard in the life sciences, and which brought the issue into focus for computer scientists as well.

Interestingly, the terminology used to describe efforts related to demonstrating, checking, or otherwise concerning the reproducibility of a piece of research is rather ill-defined to this day (Plesser 2018), riddled with misunderstandings and contradictory definitions. One of the many attempts to define reproducibility has been provided recently by the Association for Computing Machinery (ACM) as part of a new peer-review initiative called "artifact evaluation," which is poised to supplement traditional peer-review at conferences throughout computer science. The initiative distinguishes three levels of reproducibility³:

- Repeatability (Same team, Same experimental setup)

The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.

- Replicability (Different team, Same experimental setup)

The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.

- Reproducibility (Different team, Different experimental setup)

¹<http://web.stanford.edu/~vcs/Papers.html>.

²https://en.wikipedia.org/wiki/replication_crisis.

³<https://www.acm.org/publications/policies/artifact-review-badging>.

The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

Although these definitions are not necessarily the best ones, in combination with Fig. 2, they are well-suited to delineate what can and what cannot be accomplished with certain reproducibility tools in general (and TIRA in particular). Used in isolation, the term “reproducibility” encompasses all of these aspects.

In general, we distinguish efforts that target the reproducibility of an individual paper from efforts that target the reproduction of *a set of papers* all of which tackling the same (shared) task. Of course, also papers of the latter type are unique in the sense that they employ tailored methods or pursue customized solution approaches. The distinction (individual paper versus shared task papers) emphasizes that the latter address well-defined tasks which have been tackled before and for which the authors need to compare their approach to those of their predecessors. While ensuring the reproducibility of individual papers involves striving for completeness in terms of the specific experimental setups, ensuring the reproducibility of shared tasks requires some form of abstraction and unification, disregarding approach-specific details.

The efforts related to individual paper reproducibility result from the following goal: Build a kind of self-contained, reproducible publication that, even in the absence of its authors, can be used to replicate its results from scratch (Fig. 2, bottom left box). Actions that authors can *proactively* take to render their publications reproducible may be subsumed under the motto “Share all research-relevant assets.” Tools that have been proposed for this purpose include packaging software, which compiles all assets into one publication package (a single file) which in turn can be shipped and published alongside the traditional PDF (e.g., ReProZip). In this regard, asset-specific repositories and archives have been established, providing long-term preservation for certain assets (e.g., Linguistic Data Consortium, European Language Resources Association, Zenodo) as well as citability to serve as further incentive for authors to make their publications reproducible.

The efforts that can be taken by publishers, conference organizers, and third party stakeholders to further incite authors to ensure the reproducibility of their papers are reproducibility studies (Fig. 2, bottom middle box). Subject to such studies can be individual papers or sets of papers about a shared task. Note that, by nature, such studies are *reactive*; i.e., they can be done only when authors have finished their papers and are about to publish them. A fairly recent approach is to consider reproducibility within peer-reviews, either by asking reviewers of the PDF version of a paper to judge the reproducibility of its underlying assets (usually without accessing them), or by introducing an additional, dedicated review cycle called artifact evaluation: the participating authors share their assets for review and are awarded badges of honor when the reviewers find them sufficient.

Apart from these gatekeeping actions, which cannot be arbitrarily scaled and hence will affect only a small portion of papers published, third parties can

undertake a reimplementing study of either one or a set of papers. In this vein, some conferences have introduced special tracks on reproducibility to render publishing such studies easier. Another approach to analyze the reproducibility of a set of papers studying the same task are systematic reviews and meta studies, which, even without reimplementing, can reveal systematic biases in experimental setups.

Last but not least, *supportive* efforts taken by leading scientists on an established shared task include the development of effective benchmarks and gold standards (Fig. 2, bottom right box). If a benchmark is accepted and adopted by a larger portion of the community around a shared task, it ensures comparability of all papers using it, and its adoption by newcomers is often enforced via peer-reviewing, eventually rendering the benchmark “self-propagating” as the number of papers using it increases. Specific actions that support the development of benchmarks are the creation of task-specific resources, such as corpora and evaluation datasets, inquiries into measurement theory with respect to a task, developing software libraries, and not least, the organization of shared task events where participants are invited to work on a given task for which the necessary resources are provided. It can be observed that shared task events always have been instrumental in both the creation of benchmarks and the coordination of evaluation activities. For decades, especially in the human language technologies, entire conferences have been organized, some hosting dozens of shared task events at a time: TREC, CLEF, NTCIR, FIRE, SemEval, MediaEval, the KDD Cup, or CoNLL’s shared task track, to name only some of the best-known ones. TIRA has been developed within the PAN lab on digital text forensics, hosted at the CLEF conference since 2010.

Considering the above definitions of the three reproducibility levels, it becomes clear that a tool that supports proactive actions will also improve repeatability and replicability, but not reproducibility. In fact, improving the former may have adverse effects on the latter: the more assets are made available, the smaller is the need to reproduce them independently from scratch. Independent reproduction then becomes a deliberate decision instead of an everyday task for those who wish to compare their approaches to the state of the art. Again, computer science may occupy a unique position compared to other disciplines: because of the ease with which its assets can be shipped, once they are available, they may be reused without second thought, this way propagating potential biases and errors encoded in them. This characteristic is different from other empirical sciences, where sharing experimental assets is infeasible (e.g., human test subjects), so that new claims need to be independently reproduced sufficiently often before being included into the scientific canon.

TIRA fits into this picture as follows. It is a platform that has been devised as a powerful tool to support the organization of shared task events. As its most salient feature we consider the ease by which software can be submitted and, in particular, maintained for future re-execution. TIRA operationalizes blind evaluation, a paradigm that is rarely applied in empirical computer science. The term refers to an evaluation process where the authors of a to-be-evaluated piece of software cannot access the test data and hence cannot (unwittingly) optimize their algorithm against it. For this purpose TIRA implements a kind of airlock for data through which the to-be-evaluated software has to pass. The software itself is packaged within a virtual

machine and hence can be archived in working condition. Given these concepts, TIRA facilitates repeatability and replicability in first place. In addition, TIRA also supports an important reproducibility aspect, namely, the execution of a TIRA-hosted software on newly-created datasets, i.e., datasets which were not available at the time of software creation. This is ensured by providing a unified software execution interface along with harmonized dataset formats of a shared task, so that drop-in replacements become possible. At the time of writing, there are not many cloud-based evaluation systems comparable to TIRA. Collectively, these systems implement the so-called evaluation as a service paradigm. An overview of such systems has been recently compiled by Hanbury et al. (2015).

2.2 *Shared Tasks: From Run Submission to Software Submission*

When two or more groups of scientists start working independently on the same task, it literally becomes a shared task, albeit an uncoordinated one.⁴ The initiators or “inventors” of a shared task carve out the experimental setup, including evaluation resources and performance measures, and the followers may refine it or propose a new one. As a task’s followership grows, commonly accepted setups emerge and reviewers familiar with the task will hint at its proper evaluation. Eventually some stakeholder may organize an event around a shared task. The central goal of such an event is to compare latest algorithmic approaches to solving the task’s underlying problem in a controlled laboratory experiment.

A review of existing shared task events in the human language technologies revealed that such tasks have been almost unanimously organized in the same way; see Fig. 3 for an illustration. Task organizers prepare a dataset comprising problem instances, where parts of the dataset are published as training data (including the ground truth) and test data (without the ground truth) respectively. Task participants develop pieces of software that solve the task based on the training data and finally run their software on the test data. Within most shared tasks, the output of this final software run (called a *run* for short) is submitted to the organizers. The organizers, in turn, evaluate the submitted runs based on previously announced performance measures against the ground truth of the problem instances in the test data set.

To reach higher levels of automation and reproducibility, participants may submit their executable software, enabling the organizers to generate outputs by themselves, an approach we call “managed software submission.” A major obstacle to a widespread adoption of managed software submission in shared tasks is the shift of responsibility for a successful software execution. Submitted software is not necessarily free of errors—even more, experience shows that many participants submit their software prematurely, being convinced of its flawlessness. This fact

⁴The etymology of the term “shared task” is unclear; conceivably, it was coined to describe a special kind of conference track and was picked up into general use from there. We generalize the term, interpreting it literally as referring to any set of papers sharing a task.

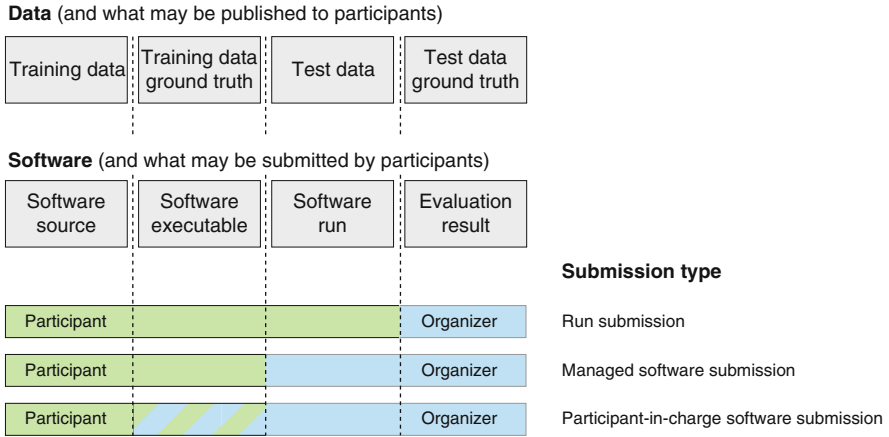


Fig. 3 From top to bottom: Task organizers develop a dataset from which certain parts are published to participants. The participants in turn develop software from which certain parts are submitted. The extent of what is published or submitted defines the submission type: run submission, managed software submission, or participant-in-charge software submission. The last submission type enables participants to submit, execute, and optimize their software, using an experiment platform (such as TIRA) provided at the organizer’s site

makes organizers unwillingly become part of the debugging process of each participant’s software, and the turnaround time to find and fix errors increases severely, especially when both parties are not working simultaneously (i.e., reside in different time zones). Failure on the part of organizers to run a submitted software, to check its output for errors of any kind (e.g., not every execution error results in a crash), and to give participants feedback in a timely manner may cause participants to miss submission deadlines. The risk of this happening is increased by the fact that many participants start working only just before a deadline, so that organizers have to handle all submissions at the same time. Besides, prolonged back-and-forth between participants and organizers caused by software errors bears a high potential for friction. As a result, organizers may come to the conclusion they have little to gain but trouble, whereas the benefits of software submissions, such as reproducibility, may be considered insufficient payback.

Our experience with managed software submissions at a shared task organized in 2013 is as follows (Gollub et al. 2013): to get the 58 pieces of software submitted running for evaluation, 1493 mails had to be exchanged in order to fix runtime errors. We postpone a more in-depth analysis of this mail exchange to Sect. 5. For now, suffice it to say that we were working hand-in-hand with participants, and that, surprisingly, participants affected by software bugs were not at all disgruntled about revisiting their software over and over again to fix them. From this experience, we derived a list of requirements (detailed in the next subsection) that a platform for shared tasks that implements software submission should fulfill. Most importantly, it must keep participants in charge of their software,

lifting responsibility from the organizers shoulders, and reducing the overhead to a bearable level for both sides.

We thus introduced a third kind of submission type, here called “participant-in-charge software submission,” providing a self-service evaluation to participants. Under this paradigm, the software is submitted to a cloud-based evaluation platform, which provides for a suitable runtime environment and manages the software. Crucially, it gives participants full control over their software and whether it works by providing runtime feedback, e.g., when run on the training data. At the request of participants, the software gets to access the test data, but that includes cutting off participant access to the software as well as the moderation of any runtime feedback by organizers. Though this approach is technically the most advanced, bringing about corresponding technical difficulties, it also comes along with appealing advantages: the software can be tested and optimized by the participants, as well as accessed, run, and archived for documentation and re-run purposes by the organizers.

2.3 Requirements Analysis

Computer science is bustling with hundreds of uncoordinated shared tasks that offer potential for future growth into shared task events. The question remains if and how the process of nurturing a shared task from its uncoordinated beginnings into a reliable evaluation activity can be formalized (and hence simplified and accelerated), and what are the requirements for a platform to support this process. Based on our experience with organizing shared task events which invited run submissions and managed software submissions, we have derived the following list of requirements.

Technological Compatibility A key requirement of a platform is its compatibility with new technologies, which emerge at a rapid pace and which have been leading to a great diversity of technology stacks and software development environments. Every developer has their own technical preferences, and minimizing compatibility constraints will help to not alienate potential users.

Setup Multiplicity Since shared tasks emerge from day to day work in a lab, those who create the first experimental setup for a given task automatically take the role of an “organizer,” defining data formats and interfaces. While their influence on successive scientists and their experimental setups is strong, one cannot expect them to get their setup right the first time around. Growing understanding of how a task can be tackled influences how it should be evaluated and dedicated scientists will not adopt a setup that does not reflect the most recent understanding. Sticking to the first setup, or excessively amplifying its importance, will result in the rejection of a platform. To avoid such conflicts, a requirement is to allow for running multiple competing experimental setups simultaneously for the same task.

Plugin Functionality Once the experimental setup of a shared task has taken shape, i.e., when all interfaces are defined, various stakeholders will develop new resources to complement existing ones. A platform for shared task creation should support the submission of all kinds of resources and, besides using them for the development of new solutions to a task, also allow third parties to analyze them. In a nutshell, the control over resources has to be relinquished to the community. Resources for a shared task include datasets, performance measures, and visualizations of datasets as well as of results. In addition, more complex tasks will require the integration of task-specific tools, e.g., in the form of external web services. A corresponding plugin architecture should hence be available.

Software Execution Layer The integration of new resources to a shared task immediately raises the question as to what performance previously evaluated approaches will achieve on them. Current shared task events typically do not collect the software of participants, so that future comparative evaluations are restricted to the same experimental setup that has been used for the original event. A platform hence should allow the reproducibility of shared task events by inviting participants to submit their software, which, of course, must be maintained and kept in working condition. Under ideal conditions, previously submitted software can be re-evaluated as soon as new resources become available for a given task.

Export and Service Layer In addition to low integration barriers both for data and software, there should also be the possibility (for all users) to export resources and to run corresponding evaluations locally instead of on the platform itself. At the same time, submitted software should be exposed via APIs if their developers wish to share it, e.g., to foster the transfer of well-performing solutions to practical use.

Governance Functionality Even if all mentioned requirements regarding technological flexibility are fulfilled, the success of a shared task platform will eventually depend on its adoption and use in practice. The people involved in a shared task must be connected with each other as well as to third parties interested in making use of the developed software. Providing a social layer can help to establish shared task governance structures, allowing for coordinated task maintenance and development. Some stakeholders may take a leading role within the community, becoming what could be called a “shared task editor.” Unlike the organizers of traditional shared task events, who typically create the complete experimental setup required, editors can ensure interface compliance of new resources, lead the community regarding certain aspects of a shared task, and organize workshops where the recent developments can be discussed. Since shared task editorship can switch between community members, the ideal shared task platform should provide basic governance functionality.

Security Executing the software of malicious participants exposes one’s infrastructure and data to exploitation. Simultaneously, submitting one’s software to a shared task platform renders it open to theft should the organizer’s systems be insecure. Moreover, hosting valuable data on such a platform invites attempts at stealing it by exploiting vulnerabilities of the platform. Therefore, a platform for shared tasks must be built with a keen eye on security from the bottom up.

3 TIRA: An Architecture for Shared Tasks

This section reports on our efforts to build a platform for shared tasks, the TIRA Integrated Research Architecture (Gollub et al. 2012a,b). TIRA has been used to organize shared task events right from the start in 2012. To date, it has handled more than a dozen shared task events and hundreds of participants and their software submissions. With the TIRA prototype, we propose solutions to most of the aforementioned requirements for shared task platforms. In particular, TIRA allows for development environment diversity, untrusted software execution, it prevents data leakage, and supports error handling by participants. Public access to TIRA's web front end is available,⁵ and its code base is shared open source.⁶ In what follows, we overview TIRA's architecture, its two most salient contributions (the datalock and blind evaluation), and give a detailed view of the user interfaces and the workflow of participants and organizers to complete a shared task.

3.1 Architecture Overview

Figure 4 gives an overview of TIRA's basic architecture. TIRA's three main components include a web service hosting user interfaces for participants and organizers, host servers for virtual machines (VM), and a storage server. All three may be running on the same physical machine, but are typically distributed across a data center. The storage server serves as a central repository for evaluation datasets, runs of participant software, and evaluation results obtained from measuring performance by comparing runs with the ground truth of a task's datasets. Participant software is encapsulated by virtual machines, and typically each participant is assigned one virtual machine per task, whereas the resources allocated to each machine may vary.

For brevity, we omit a description of the technical details and the software stack making up TIRA right now. Because of the ongoing development on the prototype, the current software stack will likely be replaced by another one at some point in the future, while the main features and functions of TIRA remain unchanged. A better way to understand TIRA is to study its most salient features, and by tracing the workflow of a typical participant and that of a typical organizer. These workflows are implemented within TIRA's web interfaces, which allow participants to remotely control software execution and to collect runtime feedback, thus eliminating the need for organizers to intervene in fixing software execution errors. At the same time, the organizers are given a number of interfaces to watch over a shared task and its participants.

⁵<http://www.tira.io>.

⁶<http://www.github.io/tira-io>.

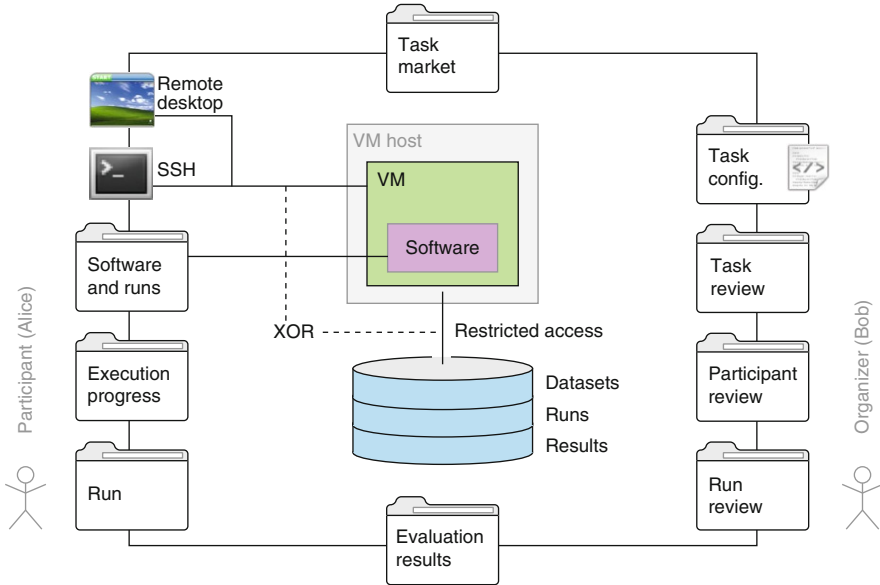


Fig. 4 TIRA's interfaces for participants (left), organizers (right), and the public (top, bottom)

3.2 The Datalock: Preventing Data Leakage

Software submitted by participants of a shared task must be treated as untrusted software, i.e., as software that may include malicious code. Virtual machines have been used for years already in all kinds of cloud platforms that allow renting servers to host and execute software, secluding every customer's software from one another. We adopt the same principles for TIRA. However, unlike in typical cloud platforms, the software deployed to the virtual machines of participants is supposed to process datasets hosted on TIRA as well, in particular the test datasets of the shared tasks. It is important that the integrity of these datasets remains intact, regardless of which software processes it, and that the test datasets do not leak.

To ensure that test data are sufficiently protected, TIRA implements what can be described as an airlock for data, the datalock. Before a software is started, the virtual machine to which it has been deployed is passed into the datalock together with the test data, where it can process the data and have its output recorded. Otherwise, all communication channels to the outside are closed or tightly moderated. To pass into the datalock, the virtual machine is cloned or snapshotted, and the copy is disconnected from the internet so that no incoming or outgoing connections are possible. The test data are then mounted read-only to the virtual machine copy. Only if a machine has been successfully moved into the datalock, is the software executed. Disconnecting the copied virtual machine from the internet while a software is executed ensures that no data can be automatically sent to an unauthorized third

party and that participants have no access to the copy of their virtual machine during software execution. After the software terminates, the output is stored in TIRA's database as a run, and the virtual machine is automatically moved out of the datalock: this boils down to either deleting the clone or resetting the virtual machine to the time of the snapshot. Deleting the copy ensures that no information about the input data remains, be it in cache, in temporary files, or in purposefully hidden files. This way, the only communication channel left is the software's output, which is kept hidden from participants by default, unless a task organizer reviews the run and concludes that it does not reveal anything important about the test data.

With some reservations, the datalock allows for shared tasks to be organized on the basis of secret, sensitive, and proprietary data. It must be conceded, though, that the security of the datalock hinges on that of the operating systems, hypervisors, and other dependent libraries of TIRA. Especially regarding extremely sensitive data, such as medical data, it may be necessary to not give participants any feedback at all, since vital information may be encoded into even the narrowest communication channel. Conversely, the fewer feedback channels remain open during software execution, the more difficult and time-consuming it becomes for participants to become aware of any software errors, since a task organizer needs to intervene first. Nevertheless, for the vast majority of shared tasks that arise from private data, the concept of the datalock constitutes a first-time opportunity for their owners to give third parties access without sharing the data itself.

3.3 Blind Evaluation for Shared Tasks

In the vast majority of evaluations carried out in empirical computer science, the experimenters have direct access to the test data and its ground truth. The validity of computer science experiments builds on trusting the experimenter to ignore the test data and its ground truth until the to-be-evaluated software is ready (we may call this approach "pseudo-blind"). Similarly, the vast majority of shared task events are organized half-blind (the test data are shared with participants but the ground truth is withheld as depicted in Fig. 3). Ideally, however, an evaluation would be conducted blind, so that a scientist has access only to the training data but not to the test data, rendering it difficult to fine-tune a given approach against the latter.

Remaining ignorant of the test data is incredibly important for an evaluation, and not doing so may have a significant impact on the validity of evaluation results. In general, one can only trust that scientists do not spoil their experiments by implicitly or explicitly exploiting their knowledge of the test data. Within shared task events, another factor comes into play: shared task events are also competitions. Dependent on its prestige, winning a shared task event may come along with a lot of visibility, so that supplying participants with the test data up front bears risks of cheating, and mistakes that spoil the ground truth.

Together with participant-in-charge software submission, the datalock may give rise to a widespread adoption of blind evaluation in shared tasks. TIRA enforces

blind evaluation by default, as long as the allowance of the number of runs against the test data is kept small by organizers.

3.4 *Life of a Participant*

From the perspective of a participant (Alice, in the following), a software submission via TIRA happens within three steps: first, deployment of the software to a given virtual machine, second, configuration of the software for remote execution, and third, remote execution of the software on the available datasets. The interfaces on the left side of Fig. 4 are used for this purpose. They put Alice in charge of deploying her software, allowing her to evaluate her software and to obtain (moderated) runtime feedback. TIRA serves as a remote control for evaluation.

TIRA encapsulates Alice's software in a virtual machine that is set up once she registers for a shared task. As depicted in Fig. 4, Alice has two ways to access her virtual machine, namely a remote desktop connection and an SSH connection. Alice retains full administrative rights inside her virtual machine, so that she can set up her preferred development environment and deploy her software. To prevent misuse, virtual machines are not allowed to communicate with each other, and, their outgoing bandwidth is limited. By default, virtual machines have only restricted access to TIRA's database, so that only the training data of each task can be read. Once a software has been successfully deployed and tested manually, participants use TIRA's web interface to complete the second and third step outlined above.

For each participant of a shared task, TIRA serves a page for the respective virtual machine, the deployed software, and the software runs. After signing in with her account for the first time, Alice can configure the execution details of her software. Figure 5 shows Alice's software control page in a state after completed configuration and a few successfully executed runs:

Virtual Machine Overview of the virtual machine, including information about the operating system, RAM, CPUs, its running state, VM host, and connectivity. The virtual machine can be turned off at the click of a button either by sending a shutdown signal to the operating system, or by powering it off. Clicking "Add Software" creates a new software panel. Alice may deploy an arbitrary number of pieces of software for the shared task on her virtual machine, e.g., to compare different paradigms or variants of an approach at solving the task. Each software can be configured individually on the software control page.

Software 1 Configuration of a software that has been deployed on the virtual machine. The software must be executable as a POSIX-compliant command. Mandatory parameters can be defined by organizers of the shared task. In this case, they include variables for input data and the output directory, and optionally for an input run (i.e., a previous run of one of Alice's pieces of software). Optionally, the working directory in which the program will be executed can be specified. Alice may adjust an existing software configuration and save its state, she may delete it, or

Virtual Machine

Operating System Ubuntu (64 bit)
RAM 4096MB
CPUs 1
State running (since 2014-06-22 09:00:00)
Sandbox state publicly accessible
Host example.com
SSH Port 44401 [open](#)
RDP Port 55501 [open](#)

[Add software](#) [Shutdown](#) [Power off](#)

Software 1

Command
The variables `$inputData` and `$inputRun` refer to the below parameters; the command must include the variable `$outputDir`. All of these variables will point to directories.

Input data

Input run
Runs on test corpora are excluded from this list.

Working directory

[Save](#) [Delete](#) [Run](#)

Evaluation

Measures precision, recall, accuracy

Input run
Evaluator runs are excluded from this list.

[Run](#)

Runs

Software	Run	Input data	Input run	Runtime	Size	Actions
evaluation	2014-06-22-12-10-00	test-data	2014-06-22-12-00-00	00:00:04	24K	i d x
software1	2014-06-22-12-00-00	test-data	none	00:01:54	2.2M	i d x
software1	2014-06-22-11-00-00	training-data	none	00:01:54	2.2M	i d x
software1	2014-06-22-10-00-00	training-data	none	00:00:30	1.1M	i d x

Fig. 5 TIRA’s web interface for participants to remotely control the execution of their software and to review their runs for a given shared task

she may proceed to execute the software. If Alice deletes a software it is not actually deleted on the server, but only hidden from view; rationale for this is to allow the reconstruction of Alice's actions for cheating prevention. The runs obtained from running a software are listed in the "Runs" panel.

Evaluation Runs an evaluation software on a given run. This is a special type of software provided by task organizers which processes an input run and outputs the results of the task's performance measures. Once Alice has finished her first successful run on a given input dataset, she uses this panel to evaluate it. The runs obtained from an evaluation software are also listed in the "Runs" panel.

Runs List of runs that have been obtained either from running a software or from running an evaluation. The table lists run details including software, timestamp, input data, input run, runtime, size on disk, and further actions that can be taken. The colorization indicates a run's status with respect to its success, where red indicates severe errors, yellow indicates warnings, green indicates complete success, and white indicates that the run has not yet been reviewed. Runs may be checked automatically for validity with the shared task's expected output format, and they may be reviewed manually by organizers. Actions that can be taken on each run include viewing more details (the blue *i*-icon), downloading it (the black arrow down), and deleting it (the red x). It is here where Alice first encounters the limitations that TIRA imposes for runs on test data: all test datasets are by default hidden from participants. TIRA prevents Alice from downloading runs on test datasets (the download action shown in gray is inactive) to prevent a malicious software from outputting the data itself instead of outputting the data that is valid for a given shared task. Even runtime and size information are initially hidden, unless a task organizer decides to unblind them.

The software control page does not display all of the aforementioned panels immediately, but only after Alice has completed the necessary steps. At first, it only shows the virtual machine panel; then, once Alice clicks on "Add Software", a software panel appears; and finally, once Alice runs her software for the first time, the evaluation panel and the runs panel are added after the run is completed. While a software is running, the software control page is replaced with the software progress monitoring page which is divided into two panels, as exemplified in Fig. 6:

Virtual Machine Just as on the software control page, the virtual machine panel shows the current state of Alice's virtual machine while the software is running. Before a software is started, the virtual machine is moved into the datalock as described above. This process may take some time, so that the intermediate states of the virtual machine are indicated in this view. The port flags indicate to Alice that access from the outside to her virtual machine has been disabled. Only if a machine has been successfully moved into the datalock, is the software executed. While a software is running, the buttons to add a software configuration panel as well as those to shutdown or power off the virtual machine are deactivated so that the running software is not interrupted accidentally. After the software terminates, the

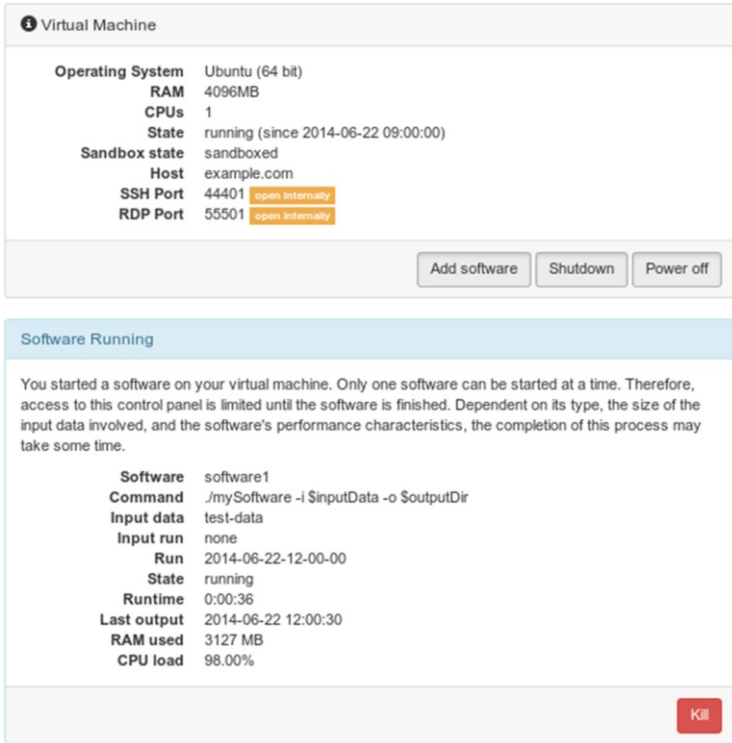


Fig. 6 TIRA’s web interface to monitor the progress of a running software

output is stored in TIRA’s database as a run, and the virtual machine is automatically moved out of the datalock.

Software Running Overview of a running software, including the software’s ID, the executed command, the parameters, the run ID, and the running state. Moreover, the current runtime, the time of the last write access to the output directory, the currently used RAM, and the CPU load are displayed and updated periodically. This way, Alice has a way of making sure her software is still working. If, for any reason, Alice wishes to kill her software before it terminates by itself, she may click on the “Kill” button. Before the software is killed, its output up to this point is stored in TIRA’s database as an incomplete run for later inspection.

After her run has completed and the virtual machine has been moved out of the datalock, Alice’s browser shows the software control page again as in Fig. 5. The new run appears in the runs table. To make sure the run was successful, Alice clicks on the *i*-icon which redirects her to a run details page for the run in question, as shown in Fig. 7a. The details shown about a run are as follows:

Overview Details about the run, including the software that was used, the run ID, parameters, whether the run can be downloaded, runtime details, its size, and the

(a)

The screenshot shows the 'Run Details' page for a software run. It is divided into three main sections: Overview, Review, and Stdout.

Overview

Software	software1
Run	2014-06-22-12-00-00
Input data	test-data
Input run	none
Downloadable	false
Runtime	00:01:54 (hh:mm:ss)
Runtime details	96.79user 8.79system 1:54.81elapsed 91%CPU (0avgtext+0avgdata 202016maxresident)k 224inputs+4160outputs (0major+14449minor)pagefaults 0swaps
Size	2.2M (154442 bytes)
Lines	0
Files	518
Directories	1

Review

Reviewer	Bob
Errors	None. This run seems to be alright.

Stdout

```
[...]t516.xml
Processing input517.xml
Writing output517.xml
Processing input518.xml
Writing output518.xml

Note: The output of software that is run against test data is shortened to
its last 100 chars.
```

File list

```
test-data/alice/2014-06-22-12-00-00/output
├── [ 90] output1.xml
├── [ 257] output2.xml
├── [ 90] output517.xml
├── [ 255] output518.xml
0 directories, 518 files
```

Download

(b)

This excerpt shows the 'Stdout' section of an evaluation run. It displays the command used to run the evaluation and the resulting precision and recall values.

Stdout

```
python shared-task-evaluation.py -i alice/2014-06-22-12-00-00/output -t
test-data -o /tmp/2014-06-22-12-10-00/output/evaluation.txt

"precision": "XXX"
"recall": "XXX"

Note: The output of evaluation runs on test corpora is blinded by default.
A task moderator will decide whether to make the results visible.
```

Stderr

Fig. 7 TIRA's web interfaces for participants to review runs. (a) Details page of a software run. (b) Excerpt of the details page of an evaluation run

numbers of lines, files, and directories found. Whether the run can be downloaded depends on whether the input data was a test dataset or not. As outlined above, runs on test datasets, by default, cannot be downloaded to foreclose data leakage. Besides the runtime, more in-depth runtime details are given, so that Alice can judge whether her software made good use of the hardware resources available to the virtual machine. For example, if she finds there are many page faults or even swaps, this indicates the software uses too much memory. The size and numbers of lines, files, and directories provide quantitative feedback to quickly verify output sanity, whereas it depends on the task which of these values is most illuminating.

Review Review of this run provided by both automatic validation and organizers. In Alice's case, an organizer reviewed the displayed run and found that it does not contain any obvious errors. In case of errors, explanations are displayed here that give insight into their nature and severity.

Stdout Standard output stream (stdout) which was recorded when executing the software. If Alice's software outputs information to stdout, it will be displayed here. However, in the case of runs on test datasets, the amount of information that is displayed can be limited. In the example, the limit is the 100 last chars of the stdout text. This limitation will prevent Alice from outputting problem instances to stdout in order to inspect them. This communication channel can be closed entirely on a per-dataset basis, for example, if confidential data has to be handled.

Stderr Standard error output stream (stderr) which was recorded when executing the software. While nothing was recorded in the example, the same filtering is applied as for the stdout stream.

File List Directory tree which displays file names and their sizes found in the run. Alice may use this information to determine whether her run has output all the files and directories that are expected, and whether their names and organization are correct.

The run details page will provide Alice with the information necessary to determine whether her remote software execution was successful. Unless the software has been executed on a test dataset, Alice may also download the run for local inspection. If she is satisfied with the run, she may proceed to evaluate it using the evaluation software. The resulting evaluation can again be inspected just like before, whereas the corresponding run details page lists the information pertaining to the evaluation software's run when receiving Alice's software run as input. Figure 7b shows an excerpt of an evaluation run details page that Alice will see. The evaluation software typically prints the evaluation results directly to stdout, however, if the evaluated software run was on a test dataset, the results are blinded by default (i.e., the performance values are replaced by "XXX"). This blinding of the evaluation results upholds blind evaluation. This way, the decision of when, if, and how the evaluation results of a given shared task are released is at the full discretion of its organizers. Moreover, just as with filtering stdout and stderr output, the organizers may adjust blinding on a per-dataset basis.

After completing her evaluation run, Alice is done; she has submitted her software to the virtual machine, made sure it works to the specifications of the shared task by running it on the available datasets and inspecting the runs for errors, and finally executed the evaluation software on her previous software runs. While Alice can now relax, it is time for the organizers of the shared task to get busy.

3.5 *Life of an Organizer*

From the perspective of an organizer (say, Bob), using TIRA to manage software submissions for a shared task can be done in three steps: first, configuration of the shared task in TIRA; second, supervision of participant progress; and third, compilation and publication of the task's evaluation results. The interfaces on the right side of Fig. 4 are used for this purpose. The configuration of a shared task is done in a configuration file. Configurable aspects include the datasets and their status (public or hidden), the evaluation software, the command line parameters required for submitted software, and various messages displayed on task-specific web pages. The web interface for task configuration is straightforward; we omit a screenshot for brevity. In terms of supervising his shared task while it is underway, Bob has three further interfaces at his disposal, an overview of participants, an overview of runs of each participant, and the run details of each participant's runs:

Task Participants (Fig. 8a) Overview of participants who have configured at least one software for Bob's shared task on their software control page, including their user name, signed in status, numbers of pieces of software that are configured, deleted, and running, and, numbers of runs that are finished, reviewed, and unreviewed. These figures give Bob an idea of whether the participants of his task are actively engaged, but it also hints about problems that may require Bob's attention. The number of deleted pieces of software may indicate that a participant has trouble setting herself up. In the case of Alice, six of seven pieces of software have been deleted, so that it may be the case that Alice had some trouble getting the software configuration right. In the case of Carol, Bob observes that her software has been running for more than six days straight, which may deviate from his expectations. Bob may contact the respective participants and offer his help. Moreover, the number of unreviewed runs indicates that some runs have not yet been checked for errors. To do so, Bob clicks on the review action (the blue eye-icon in the Actions column) to review Alice's runs; he is redirected to the participant's runs page.

Participant Runs (Fig. 8b) Overview of a participant's runs on a per-dataset basis, including the software used, run ID, input run, size, numbers of lines, files, and directories, and whether a run has been reviewed. The colorization indicates a run's status with regard to being successful, where red indicates severe errors, yellow indicates warnings, green indicates complete success, and white indicates that the run has not yet been reviewed. Unlike the runs table on Alice's software control

(a)

👤 Participants in Shared Task								
User	Signed in	Softwares	Deleted	Now Running	Runs	Reviewed	Unreviewed	Actions
Alice	yes	7	6	none	63	62	1	👁️
Carol	no	1	0	6 days, 8:37:25	4	3	1	👁️
Dan	no	1	0	none	5	0	5	👁️
Eve	no	3	1	none	16	16	0	👁️
Frank	no	3	0	none	56	56	0	👁️
Mallory	no	1	0	none	4	0	4	👁️
Oscar	no	1	0	none	4	0	4	👁️
Peggy	no	1	0	none	4	0	4	👁️
Sybil	no	3	2	none	5	5	0	👁️
Trent	no	1	0	none	4	0	4	👁️

(b)

📁 Runs of Alice on test-corpus								
Software	Run	Input run	Size	Lines	Files	Dirs	Review	Actions
evaluation	2014-06-22-12-10-00	2014-06-22-12-00-00	24K	36	1	0	todo	👁️🔍
software1	2014-06-22-12-00-00	none	2.2M	5180	518	0	done	👁️🔍
software1	2014-06-22-11-00-00	none	2.2M	5180	518	0	done	👁️🔍
software1	2014-06-22-10-00-00	none	1.1M	2590	259	0	done	👁️🔍
software1	2014-06-22-09-00-00 ^{DEL}	none	0.55M	1290	129	0	done	👁️🔍
software1	2014-06-22-08-00-00 ^{DEL}	none	1K	20	2	0	done	👁️🔍

Fig. 8 TIRA’s web interfaces for organizers to review a task’s participants. (a) Overview of a task’s participants. (b) Overview of a participant’s runs

page, this table shows figures relevant to judging a run’s success to be checked against the expectation for a given dataset: its size and the numbers of lines, files, and directories. Bob has access to all of Alice’s runs including those that have been deleted by Alice (annotated with the superscript “DEL”). Since Bob is task organizer, downloading runs is not restricted. To review the outstanding unreviewed run, Bob clicks on the corresponding review action, redirecting him to the run details page.

Run Details (Fig. 9) The run details page corresponds to that which Alice can access. It displays the same information about the run, but there are four differences. (1) it offers a review form in which Bob can enter his review, (2) the standard output streams are not filtered, (3) the output of evaluation software is not blinded, and (4) the button to download the run is always activated. Based on the complete information about the run, Bob can easily review it, which usually takes only a couple of seconds. Bob’s review consists of checking for common errors, such as missing output, extra output, output validity, as well as error messages that have

Run Details

Overview

Software	evaluation
Run	2014-06-22-12-10-00
Input data	test-data
Input run	2014-06-22-12-00-00
Downloadable	false
Runtime	00:00:04 (hh:mm:ss)
Runtime details	7.04user 14.52system 0:04.10elapsed 52%CPU (0avgtext+0avgdata 85984maxresident)k 0inputs+16outputs (0major+6224minor)pagefaults 0swaps
Size	24K (15442 bytes)
Lines	36
Files	2
Directories	0

Review

This run has not been reviewed, yet.

Reviewer Bob

Errors

- No errors
- Missing output
- Extra output
- Invalid output
- Error messages in stdout or stderr
- Other kinds of errors; please describe them in the comment below.

Comment

Stdout

```
python shared-task-evaluation.py -i alice/2014-06-22-12-00-00/output -t test-data -o /tmp/2014-06-22-12-10-00/output/evaluation.txt
```

```
"precision": "0.90081"  
"recall": "0.67283"
```

Stderr

File list

```
test-data/alice/2014-06-22-12-10-00/output/  
├── [ 246] evaluation.prototext  
└── [ 108] evaluation.txt
```

0 directories, 2 files

Fig. 9 TIRA's web interfaces for organizers to review runs


































































been printed to either standard output stream. These are the common errors that have been observed to occur frequently in previous years (Gollub et al. 2013), whereas Bob has the opportunity to write a short comment about uncommon errors he observes. Bob can supply run verification software for his task that checks runs automatically, however, at least for runs that will be used for the final evaluation results of a shared task, a quick review should be done to foreclose unforeseen errors. This reduces Bob's responsibility for the successful evaluation of Alice's software to a level similar to shared tasks that invite run submissions.

The supervision duties of task organizers cannot be entirely avoided. In shared tasks that invite run submissions, the organizers usually do not have to intervene until after the submission deadline. Only then do they learn how many participants actually submitted a run and how many of the submitted runs are valid as to the specifications of the shared task. In the extreme case, it is only after the run submission deadline, when actual examples of runs on test datasets are available, that the organizers realize that parts of the dataset or the run formats are unfit for their evaluation goals. With software submissions based on TIRA, these risks can be minimized since organizers have a chance to observe early bird participants and make adjustments as the shared task progresses. An added benefit of supervising a shared task using TIRA is that organizers learn early on how many participants actually work toward making a submission to the task, whereas with run submissions, the success or failure of a shared task in terms of number of participants will only become apparent after the run submission deadline. If Bob were to observe that only few participants start using TIRA, he may react by engaging with those who registered but did not start yet, or by advertising the task some more in the community.

Once the submission deadline has passed, and all participants have successfully evaluated their runs on the test datasets of Bob's shared task, he proceeds to reviewing the performance values and to publishing the results. For this purpose, TIRA has an overview of all evaluation runs on a per-dataset basis (see Fig. 10a):

Evaluations Results Overview of evaluation runs and the performance results obtained, including user name, software, ID of the evaluation run, ID of the software run that served as input to the evaluation run, and performance values, dependent on the measures computed by a given evaluation software. The colorization of the table cells for both run IDs corresponds to that of the run reviews mentioned above. This helps Bob to decide which are successful evaluations. All evaluation runs of all participants on a given dataset are listed. For example, there are multiple runs for participants Dan and Sybil. Bob gets to decide which of their runs are going to be published; a number of reasonable decision rules are conceivable: (1) all of them (2) the chronologically first or last successful run, (3) the run chosen by the respective participant, or (4) the best-performing run according to a given performance measure. In the Actions column, there are two publishing options, namely publication of evaluation results to the public evaluation results page (the globe icon), and publication of evaluation results to the respective participant (the person icon). As can be seen in the example, Bob has already globally published

(a)

Evaluations on <i>test-corpus</i>						
User	Software	Evaluation	Input run	Precision	Recall	Actions
Alice	software1	2014-06-22-12-10-00	2014-06-22-12-00-00	0.90081	0.67283	    
Carol	software3	2014-06-15-17-38-08	2014-06-15-17-35-38	0.85744	0.29661	    
Dan	software2 ^{CEL}	2014-06-16-17-17-21	2014-06-16-16-54-38 ^{CEL}	0.96022	0.84248	    
Dan	software3	2014-06-23-20-43-59	2014-06-23-20-17-48	0.96007	0.84511	    
Dan	software1	2014-06-16-18-03-43	2014-06-16-17-21-44	0.96243	0.83473	    
Eve	software1	2014-06-01-12-52-02	2014-06-21-05-56-23	0.82882	0.84156	    
Frank	software10	2014-06-23-13-31-42	2014-06-23-13-24-21	0.92522	0.81819	    
Mallory	software1	2014-06-20-23-28-21	2014-06-17-09-28-40	0.87171	0.91539	    
Oscar	software1	2014-06-19-00-54-42	2014-06-18-23-50-04	0.92757	0.88916	    
Peggy	software3	2014-06-22-03-36-34	2014-06-22-03-33-32	0.90032	0.80267	    
Sybil	software2	2014-06-22-02-56-09	2014-06-22-02-49-41	0.90770	0.79931	    
Sybil	software4	2014-06-22-16-55-56	2014-06-22-16-49-05	0.89179	0.80590	    
Trent	software5	2014-06-15-16-24-05	2014-06-15-15-53-28	0.86606	0.91984	    

(b)

Evaluations on <i>test-corpus</i>			
User	Precision	Recall	Runtime
Alice	0.90081	0.67283	00:04:17
Carol	0.85744	0.29661	00:00:56
Dan	0.96007	0.84511	00:19:32
Eve	0.82882	0.84156	00:05:18
Frank	0.92522	0.81819	00:02:49
Mallory	0.87171	0.91539	00:05:37
Oscar	0.92757	0.88916	00:57:15
Peggy	0.90032	0.80267	00:00:31
Trent	0.86606	0.91984	00:22:10

Fig. 10 TIRA's web interfaces for a task's evaluation results. (a) Overview of a task's evaluation results for organizers. (b) Overview of a task's *published* evaluation results

evaluation runs for all but one participant. Two of Dan's runs are published only to him, and for Sybil's two runs Bob still needs to make a decision.

The published runs appear on a public evaluation results page that can be found on TIRA alongside each shared task. Figure 10b shows the performance values of the evaluations that Bob decided to publish for his shared task. While he proceeds to announce the results to participants as well as to the scientific community, this is not necessarily the end of the story.

Shared task events are organized for a reason, and that reason is not to host an individual run-once competition, but to foster research around a problem of interest. While shared task events are sometimes organized repeatedly, at some point, they are discontinued, whereas later on there are still scientists who want to compare their approach to those of the event's participants. Based on TIRA, this will be easily

possible long after an event is over, since all the evaluation resources required to run an evaluation are hosted and kept in running state. Moreover, if new evaluation datasets appear, all previously developed approaches can be re-evaluated on the new datasets, since they are also kept in running state inside their virtual machines. This way, TIRA paves the way for ongoing, “asynchronous” evaluations around a shared task while ensuring that everyone is evaluated using the exact same experimental setup. That is, of course, as long as TIRA prevails.

4 Data Submissions: Crowdsourcing Evaluation Resources

Besides facilitating the submission of software to a given shared task (event), TIRA also opens the door to a more inclusive way of creating an experimental setup around shared tasks: there is no reason why participants should not also submit datasets and alternative performance measures. The organizer of a shared task then grows into the role of a shared task editor, instead of being personally responsible for all but the software. Data submissions for shared tasks have not been systematically studied before, so that no best practices have been established, yet. Asking a shared task event’s participants to submit data is nothing short of crowdsourcing, albeit the task of creating an evaluation resource is by comparison much more complex than average crowdsourcing tasks found in the literature. In what follows, we outline the rationale of data submissions, review important aspects of defining a data submissions task that may inform instructions to be handed out to participants, and detail two methods to evaluate submitted datasets. These are the results from our first-time experience with data submissions to one of our shared task events (Potthast et al. 2015), corresponding to the procedure we followed.

4.1 Data Submissions to Shared Tasks: A Rationale

Traditionally, the evaluation resources for a shared task event are created by its organizers—but the question remains: why? The following reasons may apply:

- *Quality control.* The success of a shared task event rests with the quality of its evaluation resources. A poorly built evaluation dataset may invalidate evaluation results, which is one of the risks of organizing shared tasks. This is why organizers have a vested interest in maintaining close control over evaluation resources, and how they are constructed.
- *Seniority.* Senior community members may have the best vantage point in order to create representative evaluation resources.
- *Access to proprietary data.* Having access to an otherwise closed data source (e.g., from a company) gives some community members an advantage over others in creating evaluation resources with a strong connection to the real world.

- *Task inventorship.* The inventor of a new task (i.e., tasks that have not been considered before), is in a unique position to create normative evaluation resources, shaping future evaluations.
- *Being first to the table.* The first one to pick up the opportunity may take the lead in constructing evaluation resources (e.g., when a known task has not been organized as a shared task event before, or, to mitigate a lack of evaluation resources).

All of the above are good reasons for an individual or a small group of scientists to organize a shared task, and, to create corresponding evaluation resources themselves. However, from reviewing dozens of shared task events that have been organized in the human language technologies, none of them are a necessary requirement (Potthast et al. 2014): shared task events are being organized using less-than-optimal datasets, by newcomers to a given research field, without involving special or proprietary data, and without inventing the task in the first place. Hence, we question the traditional connection of shared task event organization and evaluation resource construction. This connection limits the scale and diversity, and therefore the representativeness of the evaluation resources that can be created:

- *Scale.* The number of man-hours that can be invested in the construction of evaluation resources is limited by the number of organizers and their personal commitment. This limits the scale of the evaluation resources. Crowdsourcing may be employed as a means to increase scale in many situations, however, this is mostly not the case when task-specific expertise is required.
- *Diversity.* The combined task-specific capabilities of all organizers may be limited regarding the task's domain. For example, the number of languages spoken by organizers is often fairly small, whereas true representativeness across languages would require evaluation resources from at least all major language families spoken today.

By involving participants in a structured way in the construction of evaluation resources, the organizers may build on their combined expertise, man-power, and diversity. However, there is no free lunch, and outsourcing the construction of evaluation resources introduces the new organizational problem that the datasets created and submitted by third parties must be validated and evaluated for quality.

4.2 Inviting Data Submissions

When casting the instructions for data submissions to shared task event, there are a number of desiderata that participants should meet:

- *Data format compliance.* The organizers should agree on a specific data format suitable for the task in question. The format should be defined with the utmost care, since it may be impossible to fix mistakes discovered later on. Experience shows that the format of the evaluation datasets has a major effect on how

participants implement their software for a task. A dataset should comprise a set of problem instances with respect to the task, where each problem instance shall be formatted according to the specifications handed out by the organizers. To ensure compliance, the organizers should prepare a format validation tool, which allows participants to check the format of their to-be-submitted dataset, and whether it complies with the specifications. This way, participants move into the right direction from the start, and less back and forth will be necessary after a dataset has been submitted. The format validation tool should check every aspect of the required data format in order to foreclose any unintended deviation.

- *Annotation validity.* All problem instances of a dataset should comprise ground truth annotations revealing the true solution to the task in question. It goes without saying that all annotations should be valid. Datasets that do not comprise annotations are of course useless for evaluation purposes, whereas annotation validity as well as the quality and representativeness of the problem instances selected by participants determines how useful a submitted dataset will become.
- *Representative size.* The datasets submitted should be of sufficient size, so that dividing them into training and test datasets can be done without sacrificing representativeness, and so that evaluations conducted on the basis of the resulting test datasets are meaningful and not prone to noise.
- *Choice of data source.* The choice of a data source should be left up to participants, and should open the possibility of using manually created data either from the real world or by asking human test subjects to emulate problem instances, as well as automatically generated data based on a computer simulation of problem instances for the task at hand.
- *Copyright and sensitive data.* Participants must ensure that they have the usage rights of the data, for transferring usage rights to the organizers of the shared task event, and for allowing the organizers to transfer usage rights to other participants. The data must further be compliant with privacy laws and ethically innocuous. Dependent on the task at hand and what the organizers of the event desire, accepting confidential or otherwise sensitive data may still be possible by exploiting the datalock: by inviting software submissions to TIRA, the organizers may ensure that the sensitive data does not leak to participants, running submitted software at their site against the submitted datasets.

4.3 Evaluating Data Submissions

The construction of new evaluation datasets must be done with the utmost care, since datasets are barely double-checked or questioned again once they have been accepted as authoritative. This presents the organizers who invite data submissions with the new challenge of evaluating submitted datasets, where the evaluation of a dataset should aim at establishing its validity. In general, the organizers of a shared task event that invites data submissions should take care not to advertise submitted

datasets as valid unless they are, since such an endorsement may carry a lot of weight in a shared task's community.

Unlike shared task events that invite algorithmic contributions, the validity of a dataset typically can not be established via an automatically computed performance measure, but requires manual reviewing effort. Though peer-review is one of the traditional means of the scientific community to check and ensure quality, data submissions introduce new obstacles for the following reasons:

- *Dataset size.* Datasets for shared tasks tend to be huge, which renders individual reviewers incapable of reviewing them all. Here, the selection of a statistically representative subset may alleviate the problem, allowing for an estimation of the total amount of errors or other quality issues in a given dataset.
- *Assessment difficulty.* Even if the ground truth of a dataset is revealed, it may not be enough to easily understand and follow the construction principles of a dataset. Additional tools may be required to review problem instances at scale; in some cases, these tools need to solve the task's underlying problem, e.g., to properly visualize problem instances, whereas, without visualization, the review time per problem instance may be prohibitively long.
- *Reviewer bias.* Given a certain assessment difficulty for problem instances, even if the ground truth is revealed, reviewers may be biased to favor easy decisions over difficult ones.
- *Curse of variety.* While shared task events typically address very clear-cut problems, the number of application domains where the task in question occurs may be huge. In these situations, it is unlikely that the reviewers available possess all the required knowledge, abilities, and experience to review and judge a given dataset with confidence.
- *Lack of motivation.* While it is fun and motivating to create a new evaluation resource, reviewing those of others is less so. Reviewers in shared task events that invite data submissions may therefore feel less inclined to invest their time in reviewing other participants' contributions.
- *Privacy concerns.* Some reviewers may feel uncomfortable when passing open judgment on their peers' work for fear of repercussions, especially when they find datasets to be sub-standard. However, an open discussion of the quality of evaluation resources of all kinds is an important prerequisite for progress.

Nevertheless, as long as no third party, impartial reviewers are at hand, as part of their participation, all participants who submit a dataset to a shared task event should be compelled to also peer-review the datasets submitted by other participants. The reviewers may be instructed as follows:

The peer-review is about dataset validity, i.e. the quality and realism of the problem instances. Conducting the peer-review includes:

- *Manual* review of as many examples as possible from all datasets
- Making observations about how the dataset has been constructed
- Making observations about potential quality problems or errors
- Making observations on the realism of each dataset's problem instances

- Writing about your observations in your notebook (make sure to refer to examples from the datasets for your findings).

Handing out the complete submitted datasets for peer-review, however, is out of the question, since this would defeat the purpose of subsequent blind evaluations by revealing the ground truth prematurely. Here, the organizers serve as mediators, splitting submitted datasets into training and test datasets, and handing out only the training portion for peer-review. Obviously, participants who submitted a dataset and who also submitted a software tackling the task in question cannot be subject to blind evaluation on their own dataset, since they possess a copy of the portion of their dataset used as test data. Such conflicts of interest should be highlighted.

Finally, when a shared task event has previously invited software submissions, this creates ample opportunity to re-evaluate the existing software on the submitted datasets. It becomes possible to evaluate submitted datasets in terms of their difficulty: the performance scores of existing software on submitted datasets, when compared to their respective scores on established datasets, allow for a relative assessment of dataset difficulty. Otherwise, the organizers should set up a baseline software for the task and run that against submitted datasets to allow for a relative comparison among them.

5 User Experience and Usage Scenarios

Throughout the years of operating TIRA as a prototypical shared task platform, hosting more than a dozen shared tasks since 2012 until the time of writing, and handling hundreds of software submissions to them, we have gained substantial experience and acquired important insight. To test its limits, we specifically exploited TIRA's capabilities to organize shared task events with advanced experimental setups some of which would not have been feasible otherwise. These usage scenarios and the resulting feedback about user experience inform our ongoing development of TIRA from an already robust prototype to a fully-fledged shared task platform. In what follows, we candidly report successes and failures.

5.1 *User Feedback on the Prototype*

We received a great deal of positive user feedback about TIRA from shared task participants as well as their organizers. This has recently been substantiated by the organizers of the CoNLL 2017 shared task, who conducted a user survey; 14 of the 36 participants responded and the results can be seen in Fig. 11. The overwhelming majority of participants, 71%, fully support TIRA and its cause, whereas the remainder ask for improved debugging facilities, significant further improvements, or question the necessity of evaluation platforms altogether. Given

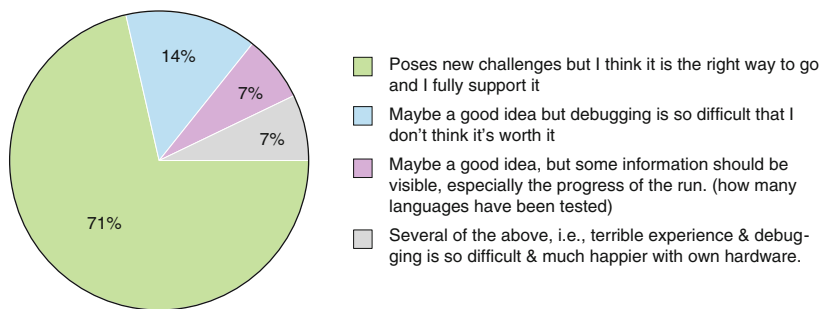


Fig. 11 Feedback from participants of the CoNLL 2017 shared task

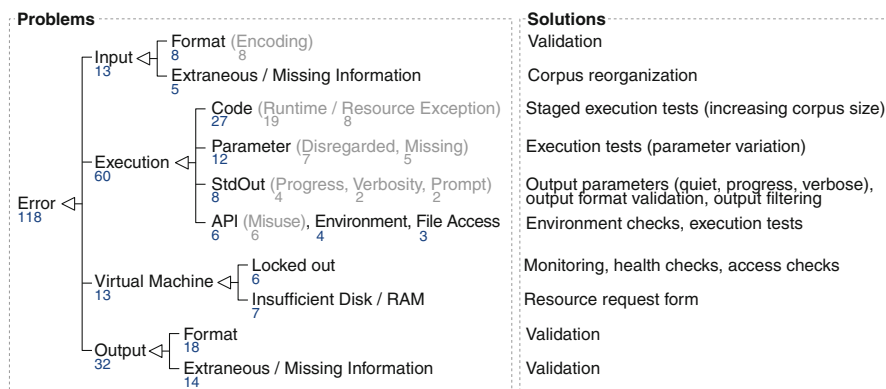


Fig. 12 Taxonomy of 118 problems that occurred during a busy shared task event that invited software submissions along with technical solutions that identify them automatically. The numbers indicate the amount of errors within each category

the fact that TIRA, in its current form, is an early prototype, and the fact that participants in shared task events such as the one hosted by the CoNLL conference expect high performance of the tools they are supposed to use—after all, research on the task is their primary goal—it is a success that the TIRA prototype achieves this much positive feedback.

To gain more insights into what, precisely, are the errors experienced by TIRA’s users, and how the platform can be further developed to mitigate them, we analyzed the mail correspondence of one of our earlier shared task events (Gollub et al. 2013): 1493 mails were exchanged within 392 conversations, discussing 118 errors. The number of teams experiencing at least one error is 39 from a total of 46, whereas 26 teams experienced at least two errors and one unlucky team 10. The identification of errors and the subsequent discussions induced a significant amount of manual workload. Sometimes, more than one round-trip was necessary to resolve an error. To get a better idea of what kinds of errors occurred and how they can be prevented in the future, we organized them into a taxonomy depicted in; Fig. 12.

In general, input and output errors can be observed in traditional run submission tasks as well, whereas execution errors and virtual machine errors are exclusive to software submission tasks. While the former can be easily identified or prevented by providing format validation and simplifying dataset organization, the latter require more intricate solutions or cannot be identified automatically at all. Since half of all errors are execution errors, the work overhead for organizers is minimized when participants perform execution tests themselves via the web frontend on small-scale trial and training datasets.

5.2 *User Retention and Conversion Rate*

The acceptance of shared task platforms also depends on implicit user feedback, namely user retention and conversion rate. The former refers to the retention of users when a shared task event switches from run submissions to software submissions, and the latter to the conversion of registered users to users who submit a software.

User Retention when Switching to Software Submissions Given the fact that almost all shared task events today invite run submissions, a barrier to market entry of a shared task platform that solicits software submissions comes in the form of lock-in-effects, where (1) the organizers of successful shared task events are unwilling to switch to another paradigm, since the one at hand works well, and where (2) the organizers of new shared task events follow the example of the majority, who employ run submissions. From those organizers to whom we talked and who considered switching to software submission, they feared that either participants would shy away from the additional overhead, or that organizers would be overwhelmed by it.

Since TIRA was developed within our own shared task series PAN, hosted at the CLEF conference, naturally we employed it there first, starting 2012 with one shared task, and switching all shared tasks to software submission as of 2013. Table 1 shows the numbers of registrations, run/software submissions, and notebook paper submissions for PAN before and after the switch. PAN was growing at the time, and as it turned out, TIRA did not harm the growth of PAN’s shared tasks in terms of participants. Neither of the shared tasks have failed because of TIRA, nor have participation rates dropped compared to the previous “traditional” submission type.

Another success story was the switch of the well-known shared task hosted annually at the CoNLL conference. The CoNLL shared task series has been running since 2000, exchanging the organizer team every 1–2 years. This shared task series is quite renowned and attracts many participants also from well-known institutions. The ACL special interest group of natural language learning (SIGNLL) approached us to inquire about TIRA, and recommended the use of the prototype to implement software submissions for the 2015 edition to its organizers. Ever since, we offer TIRA to the organizers of CoNLL’s shared task events and assist them with the setup. Again, when compared to the participation rates of the many previously

Table 1 Key figures of the PAN workshop series, hosted at the conferences SEPLN and CLEF

Statistics	SEPLN	CLEF							
	2009	2010	2011	2012	2013	2014	2015	2016	2017
Registrations	21	53	52	68	110	103	148	143	191
Runs/software	14	27	27	48	58	57	54	37	34
Notebooks	11	22	22	34	47	36	52	29	30

Run submission (2010-2012) | Software submission (2013-2017)

running shared tasks at CoNLL, we did not observe any significant impact after the switch.

User Retention Within Shared Tasks Hardly any research has been carried out to date about what makes a shared task (event) successful, or what are success indicators. The number of participants a shared task attracts may be such an indicator, where many consider more participants indicative of more success. This view, however, can easily be refuted, since even a single participant can render a shared task successful, if he or she contributes a groundbreaking approach (i.e., “quality over quantity”). Nevertheless, the number of registrations for a shared task event is indicative of initial interest from the community, whereas the conversion rate from registration to submission is typically significantly less than 100% (as can also be seen in Table 1). Regarding software submissions in general, and TIRA in particular, the question arises whether and how they affect the conversion rate. For lack of data that allows for a statistical analysis, we resort to our experience with organizing the WSDM Cup shared task, which makes for an illuminating showcase in this respect.

The WSDM Cup 2017, organized as part of the ACM WSDM conference, had two tasks for which a total of 140 teams registered, 95 of which ticked the box for participation in the vandalism detection task (multiple selections allowed). This is a rather high number compared to other shared task events. We attribute this success to the facts that the WSDM conference is an A-ranked conference, giving the WSDM Cup a high visibility, that the vandalism detection task was featured on Slashdot,⁷ and that we attracted sponsorship from Adobe, which allowed us to award cash prizes to the three winning participants of each task. However, only 35 registered participants actually engaged when being asked for their operating system preferences for their virtual machine on TIRA, 14 of which managed to produce at least one run, whereas the remainder never used their assigned virtual machines at all. In the end, 5 teams made a successful submission to the vandalism detection task by running their software without errors on the test dataset. By contrast, from the 51 registered teams for the second task, triple scoring, 21 successfully submitted their software.

⁷<https://developers.slashdot.org/story/16/09/10/1811237>.

Why did so many participants drop out on the vandalism task? We believe that the comparably huge size of the dataset as well as difficulties in setting up their approach on our evaluation platform are part of the reason: each approach had to process gigabytes of data, implement a client-server architecture for this purpose, and it all had to be deployed on a remote virtual machine. The requirement to submit working software, however, may not have been the main cause since the conversion rate of the companion task was much higher. Rather, the combination of dataset size, real-time client-server processing environment, and remote deployment is a likely cause. Note that the vandalism detection task itself demanded this setup, since otherwise it would have been easy to cheat, which is a problem when cash prizes are involved. Finally, the provided baseline systems were already competitive, so that the failure to improve upon them may have caused additional dropouts.

The WSDM Cup taught us an important lesson about the opportunities and limitations of shared task events in general and about evaluation platforms and rule enforcement in particular. On the one hand, competitions like this are important to rally followers for a given task and to create standardized benchmarks. On the other hand, shared task events are constrained to a relatively short period of time and create a competitive environment between teams. I.e., it becomes important to implement a good trade-off in the evaluation setup in order to prevent potential cheating and data leaks, while, at the same time, placing low barriers on the submission procedure. Moreover, there definitely is a trade-off between strict rule enforcement on the one side and scientific progress on the other. For example, only two teams had made a successful submission by the original deadline, while other teams were still struggling with running their approaches. In this case, we erred on the side of scientific progress and accepted submissions up to 8 days past the official deadline, instead of enforcing it. This caused some discussion about the deadline's extensions fairness, where we had to defend the opportunity of more scientific progress over the competitive mindsets of some participants.

Altogether, we do not believe that the low conversion rate of the vandalism detection task was primarily caused by TIRA, but that the task's advanced evaluation setup as a whole did. Otherwise, the companion task would also have had a lower conversion rate. That does not mean that advanced task setups cannot be successful using TIRA. On the contrary, the following section outlines a number of examples where task complexity did not negatively affect conversion rates.

5.3 Advanced Shared Task Setups

One of the most exciting opportunities that the TIRA prototype offers is to explore new experimental setups of shared tasks. Apart from the aforementioned setup for the WSDM Cup, which involved stream analysis, we have successfully implemented a number of innovative setups that can be seen as testimony to the potential that shared task platforms offer in general. Moreover, they demonstrate the versatility of TIRA in being adapted to different needs.

Cross-Event Evaluation One of the primary goals of inviting software submissions in a shared task event is to make re-evaluations of the submitted software on different datasets possible. We grasped at this opportunity immediately after we organized a shared task event for the second time in a row on TIRA, demonstrating this possibility by cross-evaluating software from the previous edition on the then-current evaluation datasets and vice versa. This way, participating in one shared task event corresponds to participating in all of them past, present, and future. Moreover, if a participant submits versions of their software in different years, this makes it possible to track performance improvements. A complete discussion of the results is out of the scope of this section, however, they can be found in Potthast et al. (2013). Cross-year evaluations have become the norm since TIRA is in use for tasks that are organized repeatedly.

Incorporating External Web Services When moved into the datalock, TIRA prevents virtual machines from accessing the internet. For some task setups, however, this may be too limiting. For example, an API offered by a third party, which cannot be installed locally inside a virtual machine, may be instrumental to solving a task. The datalock is implemented using a standard firewall, which prevents a virtual machine's virtual network interface from accessing the internet. Naturally, the configuration of the firewall allows for relaxing these access restrictions partially for pre-specified IPs or domains. TIRA implements whitelisting and blacklisting of host names and their associated IP addresses, thereby giving task organizers the liberty to decide whether any, or even just individual participants are allowed to access a given web service.

As part of a series of task events on plagiarism detection, we offered the shared task source retrieval, which is about retrieving a candidate source document from the web for a given suspicious document using a web search engine's API. For sake of realism, we set up a fully-fledged search engine for the ClueWeb corpus, called ChatNoir (Potthast et al. 2012). Given the size of the ClueWeb, we were not able to host more than one instance, let alone one in each participants' virtual machine, so that all submitted software from participants was required to access the search engine. We hence adjusted the datalock firewall setting to give virtual machines access to the host of the web server hosting the ChatNoir's API.

Shared Task Scale and Resource Allocation The CoNLL 2017 shared task served as a test of scalability of the evaluation as a service paradigm in general as well as that of TIRA in particular (Zeman et al. 2017). The allocation of an appropriate amount of computing resources (especially CPUs and RAM, whereas disk space is cheap enough) to each participant proved to be difficult, since minimal requirements were unknown. When asked, participants typically request liberal amounts of resources, just to be on the safe side, whereas assigning too much up front would not be economical nor scale well. We hence applied a least commitment strategy with an initial assignment of 1 CPU and 4 GB RAM. More resources were granted on request, the limit being the size of the underlying hardware. When it comes to exploiting available resources, a lot depends on programming prowess, whereas more resources do not necessarily translate into better performance. This is best

exemplified by the fact that with 4 CPUs and 16 GB RAM, the winning team Stanford used only a quarter the amount of resources of the second and third winners, respectively. The team in fourth place was even more frugal, getting by with 1 CPU and 8 GB RAM. All of the aforementioned teams' approaches are within the same ballpark performance, showing that the amount of resources available is no indicator of success at a shared task. Arguably, this may not hold in all conceivable cases.

Adversarial Shared Tasks Many tasks that are studied in computer science have an adversarial companion task, where solving one means defeating the solutions for the other. For example, the task of author identification, where given a document, the question to be answered is who wrote it, has its counter in the task of author obfuscation, where given a document, rewrite it so that its author cannot be identified. Here, the performance of an author identifier can only be appreciated if it also defeats all obfuscators available, and vice versa. However, the research community around author identification has never carried out extensive experiments to establish the capabilities of either technology versus the other. This is due to the fact that a serious experiment for a piece of either kind of software requires the procurement of working implementations of *all* existing technologies for the other. Given the amount of author identification technologies that have been proposed in its more than 100 years history, this is an arduous task that can hardly be tackled by any individual or small team of scientists.

The game changes, however, when software submissions come into play. Based on three years worth of software submissions to PAN's author identification tasks (a total of 44 author identifiers representing the state of the art), we have organized for the first time an author obfuscation task, where the goal was to beat each and every one of the identifiers (Potthast et al. 2016). To the best of our knowledge, an evaluation involving adversarial technologies at this scale has hardly any precedent in computer science, if at all, especially when taking into account the low workload at our end. By extension, this demonstrates that, based on a shared task platform like TIRA, compatible or adversarial shared tasks can be composed into pipelines involving even more than just two tasks, thereby opening the door to systematic evaluations of technologies solving very complex tasks.

Data Submissions Shared task platforms render data submissions feasible, since the datasets can be immediately evaluated against previously submitted pieces of software for the task in question. One of the longest-running shared tasks at PAN has been text alignment for plagiarism detection. Given that a stable community formed around this task in previous years, and that the data format has not changed throughout the years, we felt confident to experiment with this task and to switch from algorithm development to data submissions. We cast the task to construct an evaluation dataset as follows:

- *Dataset collection.* Gather real-world instances of text reuse or plagiarism, and annotate them.

- *Dataset generation.* Given pairs of documents, generate passages of reused or plagiarized text between them. Apply a means of obfuscation of your choosing.

The task definition has been kept as open as possible, imposing no particular restrictions on the way in which participants approach this task, which languages they consider, or which kinds of plagiarism obfuscation they collect or generate. In particular, the task definition highlights the two possible avenues of dataset construction, namely manual collection, and automatic construction. To ensure compatibility with each other and with previous datasets, however, the format of all submitted datasets had to conform with that of the existing datasets used in previous years. By fixing the dataset format, future editions of the text alignment task may build on the evaluation resources created within the data submission task without further effort, and the pieces of software that have been submitted in previous editions of the text alignment task, available on the TIRA platform, have been re-evaluated on the new datasets. In our case, more than 31 text alignment approaches have been submitted since 2012. To ensure compatibility, we handed out a dataset validation tool that checked all format restrictions. A total of eight datasets have been submitted, offering great variety of languages and data sources. Our approach at validating data submissions for shared tasks followed the procedures outlined in Sect. 4: all participants who submit a dataset have been asked to peer-review the datasets of all other participants, and, all 31 pieces of software that have been submitted to previous editions of our shared task on text alignment were evaluated against the submitted datasets.

We have observed all of the obstacles to peer-review outlined in Sect. 4: some submitted datasets were huge, comprising thousands of generated plagiarism cases; reviewing pairs of entire text documents up to dozens of pages long, and comparing plagiarism cases that may be extremely obfuscated is a laborious task, especially when no tools are around to help; some submitted datasets have been constructed in languages that none of the reviewers speak, except for those who constructed the dataset; and some of the invited reviewers apparently lacked the motivation to actually conduct a review in a useful manner. Nevertheless, the peer-review alongside performance characteristics obtained from the re-evaluation of the 31 plagiarism detectors submitted to the text alignment task in previous years on each submitted dataset gives us confidence that the submitted datasets are reasonably suited to serve as evaluation datasets in the future, significantly increasing the diversity of plagiarism detection corpora available today.

Altogether, as a result of all of these experiments with the design of shared task setups, and because of the reasonable success of the associated shared task events from which ample scientific progress could be extracted, we are happy to say that the TIRA prototype implementation forms a solid foundation to build on in the future. Nevertheless, there are still many avenues of innovation to be explored, and correspondingly many new features need to be developed, let alone the necessary improvement of user experience as well as meeting all requirements outlined at the outset of the chapter.

6 Conclusion

The TIRA Integrated Research Architecture offers one of the first and currently the most fully developed platform for shared tasks in the cloud under the evaluation as a service paradigm. Our long-term experience with operating the TIRA prototype, applying it in practice at the PAN workshop at the CLEF conference, at the shared task of the CoNLL conference, and at various other shared task events, has placed TIRA in a unique position to grow, transcending the human language technologies and transferring the concept of shared task evaluations to other branches of computer science. To facilitate this opportunity, TIRA has been released open source, inviting everyone to contribute to its development. This way, we hope to encourage more shared tasks in computer science to gain followership and eventually grow into shared task events. As a platform, TIRA has proven itself, handling more than a dozen shared tasks, some many years in a row, and hundreds of software submissions since its first use in 2012. If successful on a larger scale, TIRA may serve to improve the reproducibility of computer science as a whole.

References

- Claerbout J, Karrenbach M (1992) Electronic documents give reproducible research a new meaning, pp 601–604. <https://doi.org/10.1190/1.1822162>
- Donoho D, Maleki A, Rahman I, Shahram M, Stodden V (2009) Reproducible research in computational harmonic analysis. *Comput Sci Eng* 11:8–18. <https://doi.org/10.1109/MCSE.2009.15>
- Fokkens A, van Erp M, Postma M, Pedersen T, Vossen P, Freire N (2013) Offspring from reproduction problems: What replication failure teaches us. In: *Proceedings of the 51st annual meeting of the association for computational linguistics (Long papers)*, vol 1. Association for Computational Linguistics, Sofia, pp 1691–1701. <http://www.aclweb.org/anthology/P13-1166>
- Freire J, Fuhr N, Rauber A (2016) Reproducibility of data-oriented experiments in e-science (dagstuhl seminar 16041). *Dagstuhl Rep.* 6(1):108–159. <https://doi.org/10.4230/DagRep.6.1.108>
- Gollub T, Stein B, Burrows S (2012a) Ousting Ivory tower research: towards a web framework for providing experiments as a service. In: Hersh B, Callan J, Maarek Y, Sanderson M (eds) 35th international ACM conference on research and development in information retrieval (SIGIR 2012). ACM, New York, pp 1125–1126. <https://doi.org/10.1145/2348283.2348501>
- Gollub T, Stein B, Burrows S, Hoppe D (2012b) TIRA: configuring, executing, and disseminating information retrieval experiments. In: Tjoa A, Liddle S, Schewe KD, Zhou X (eds) 9th international workshop on text-based information retrieval (TIR 2012) at DEXA. IEEE, Los Alamitos, pp 151–155. <https://doi.org/10.1109/DEXA.2012.55>
- Gollub T, Potthast M, Beyer A, Busse M, Rangel Pardo F, Rosso P, Stamatatos E, Stenno B (2013) Recent trends in digital text forensics and its evaluation—plagiarism detection, author identification, and author profiling. In: Forner P, Müller H, Paredes R, Rosso P, Stein B (eds) *Information access evaluation meets multilinguality, multimodality, and visualization. Proceedings of the fourth international conference of the clef initiative (CLEF 2013). Lecture notes in computer science (LNCS)*, vol 8138, Springer, Heidelberg, pp 282–302

- Hanbury A, Müller H, Balog K, Brodt T, Cormack G, Eggel I, Gollub T, Hopfgartner F, Kalpathy-Cramer J, Kando N, Krithara A, Lin J, Mercer S, Potthast M (2015) Evaluation-as-a-service: overview and outlook. <http://arxiv.org/abs/1512.07454>
- Pedersen T (2008) Empiricism is not a matter of faith. *Comput Linguist* 34(3):465–470 <https://doi.org/10.1162/coli.2008.34.3.465>
- Plesser HE (2018) Reproducibility vs. replicability: a brief history of a confused terminology. *Frontiers in Neuroinformatics* 11:76. <https://doi.org/10.3389/fninf.2017.00076>
- Potthast M, Hagen M, Stein B, Graßegger J, Michel M, Tippmann M, Welsch C (2012) ChatNoir: A search engine for the ClueWeb09 corpus. In: Hersh B, Callan J, Maarek Y, Sanderson M (eds) 35th international ACM conference on research and development in information retrieval (SIGIR 2012). ACM, New York, p 1004. <https://doi.org/10.1145/2348283.2348429>
- Potthast M, Gollub T, Hagen M, Tippmann M, Kiesel J, Rosso P, Stamatatos E, Stein B (2013) Overview of the 5th international competition on plagiarism detection. In: Forner P, Navigli R, Tufis D (eds) Working notes papers of the CLEF 2013 evaluation labs. <http://www.clef-initiative.eu/publication/working-notes>
- Potthast M, Gollub T, Rangel Pardo F, Rosso P, Stamatatos E, Stein B (2014) Improving the reproducibility of PAN's shared tasks: plagiarism detection, author identification, and author profiling. In: Kanoulas E, Lupu M, Clough P, Sanderson M, Hall M, Hanbury A, Toms E (eds) Information access evaluation—multilinguality, multimodality, and interaction. Proceedings of the fifth international conference of the CLEF initiative (CLEF 2014). Lecture notes in computer science (LNCS), vol 8685, Springer, Heidelberg, pp 268–299
- Potthast M, Göring S, Rosso P, Stein B (2015) Towards data submissions for shared tasks: first experiences for the task of text alignment. In: Working notes papers of the CLEF 2015 evaluation labs, CLEF and CEUR-WS.org, CEUR workshop proceedings. <http://www.clef-initiative.eu/publication/working-notes>
- Potthast M, Hagen M, Stein B (2016) Author obfuscation: attacking the state of the art in authorship verification. In: Working notes papers of the CLEF 2016 evaluation labs, CLEF and CEUR-WS.org, CEUR workshop proceedings, vol 1609. <http://ceur-ws.org/Vol-1609/>
- Stodden V (2010) The scientific method in practice: reproducibility in the computational sciences. Tech. Rep. MIT Sloan Research Paper No. 4773-10. <https://doi.org/10.2139/ssrn.1550193>
- Zeman D, Popel M, Straka M, Hajic J, Nivre J, Ginter F, Luotolahti J, Pyysalo S, Petrov S, Potthast M, Tyers F, Badmaeva E, Gokirmak M, Nedoluzhko A, Cinkova S, Hajic jr J, Hlavacova J, Kettnerová V, Uresova Z, Kanerva J, Ojala S, Missilä A, Manning C, Schuster S, Reddy S, Taji D, Habash N, Leung H, de Marneffe MC, Sanguinetti M, Simi M, Kanayama H, de Paiva V, Droганова K, Martínez Alonso H, Çöltekin Ç, Sulubacak U, Uszkoreit H, Macketanz V, Burchardt A, Harris K, Marheinecke K, Rehm G, Kayadelen T, Attia M, Elkahky A, Yu Z, Pitler E, Lertpradit S, Mandl M, Kirchner J, Fernandez Alcalde H, Strnadová J, Banerjee E, Manurung R, Stella A, Shimada A, Kwak S, Mendonca G, Lando T, Nitisaroj R, Li J (2017) CoNLL 2017 shared task: multilingual parsing from raw text to universal dependencies. In: Proceedings of the CoNLL 2017 shared task: multilingual parsing from raw text to universal dependencies. Association for Computational Linguistics, pp 1–19. <https://doi.org/10.18653/v1/K17-3001>