CODABENCH: FLEXIBLE, EASY-TO-USE AND REPRODUCIBLE META-BENCHMARK PLATFORM

ACCEPTED BY PATTERNS CELL PRESS

Zhen Xu* **Sergio Escalera** **Adrien Pavão** **Magali Richard**

Wei-Wei Tu **Quanming Yao** **Huan Zhao** **Isabelle Guyon**

SUMMARY

Obtaining standardized benchmark of computational methods is a major issue in data science communities. Dedicated frameworks enabling fair benchmarking in a unified environment are yet to be developed. Here we introduce Codabench, a meta-benchmark platform that is open-sourced and community driven for benchmarking algorithms or software agents versus datasets or tasks. A public instance of Codabench `https://www.codabench.org/` is open to everyone, free of charge, and allows benchmark organizers to compare fairly submissions, under the same setting (software, hardware, data, algorithms), with custom protocols and data formats. Codabench has unique features facilitating easy organization of flexible and reproducible benchmarks, such as the possibility of re-using templates of benchmarks, and supplying compute resources on-demand. Codabench has been used internally and externally on various applications, receiving more than 130 users and 2500 submissions. As illustrative use cases, we introduce four diverse benchmarks covering Graph Machine Learning, Cancer Heterogeneity, Clinical Diagnosis and Reinforcement Learning.

Keywords Machine Learning · Data Science · Benchmark platform · Reproducibility · Competitions

1 Introduction

The methodology of unbiased algorithm evaluation is crucial for machine learning, and has recently received renewed attention in all data science scientific communities. Often, researchers have difficulties understanding which dataset to choose for a fair evaluation, with which metrics, under which software/hardware configurations, and on which platforms. The concept of benchmark itself is not well standardized and includes many settings. For instance, the following may be referred to as a benchmark: a set of datasets; a set of artificial tasks; a set of algorithms; one or several dataset(s) coupled with reference baseline algorithms; a package for fast prototyping algorithms for a specific task; a hub for compilation of related algorithm implementations. In addition, many benchmarks often integrate new progresses by manual verification instead of automatic submission and execution, which delays the benchmark update and requires extra human efforts.

Typical examples of existing frameworks addressing such needs are inventoried in Table 1, including competition platforms, repository hubs and domain specific benchmarks. Firstly, competition platforms focus on the participants and

*Additional authorship information is as follows:
Corresponding authors: Zhen Xu (`xuzhen@4paradigm.com`) and Isabelle Guyon (`guyon@chalearn.org`).
Lead contact: Zhen Xu (`xuzhen@4paradigm.com`).
Authors' order: The other authors are ordered alphabetically.
Affiliation: Zhen Xu, Wei-Wei Tu, Huan Zhao are with 4Paradigm, Beijing 100085, China; Sergio Escalera is with Universitat de Barcelona, Computer Vision Center, Barcelona 08007, Spain; Adrien Pavão and Isabelle Guyon are with LISN/CNRS/INRIA, University Paris-Saclay, Gif-sur-Yvette 91190, France; Magali Richard is with University Grenoble Alpes, CNRS, UMR 5525, VetAgro Sup, Grenoble INP, TIMC, Grenoble 38000, France; Quanming Yao is with Tsinghua University, Beijing 100084, China; Isabelle Guyon is also with ChaLearn, CA, USA.

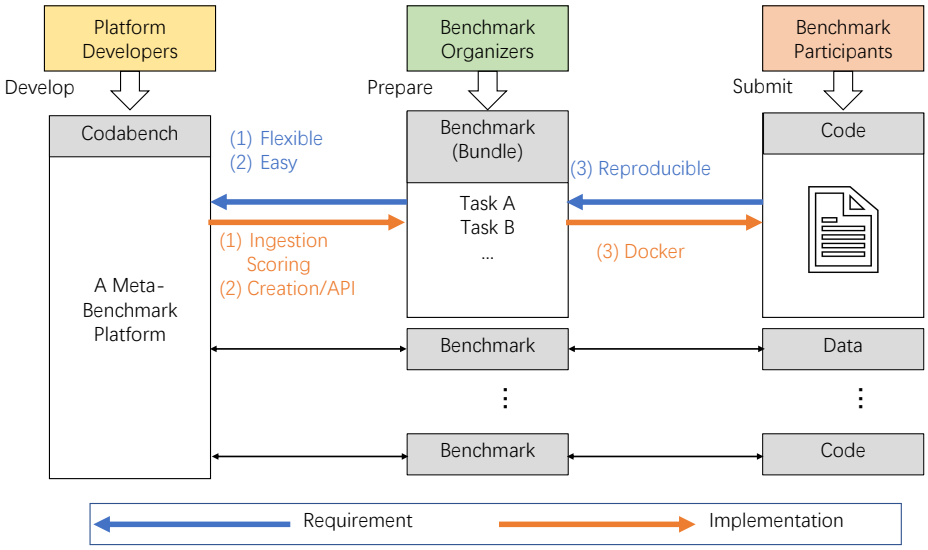


Figure 1: **Overview of Codabench.** A meta-benchmark platform has three types of contributors: platform developers (in yellow), benchmark organizers (in green) and benchmark participants (in red). *Codabench* is at the meta level to support diverse benchmarks. Each benchmark is implemented by a benchmark bundle that contains one or more tasks.

provide limited support for organizing general tasks. Famous platforms like Kaggle[2], Tianchi[3], CodaLab[4] organize many data science challenges attracting a large number of participants. However, the platform providers retain some control: the organizers do not have full flexibility and control over their competitions. Thus, the experience for organizers are not enjoyable. A comparison between competitions and benchmarks is given in Table S1 in the supplementary. Secondly, repository hubs such as UCI repository[5], OpenML[19] and PapersWithCode[6], also play an important role for benchmarks and research. They collect large amount of datasets, methods, and results from academic papers, but reproducibility by running code in given containers (or similar ways) is not guaranteed. Besides the above-mentioned platforms, many domain specific benchmarks exist, e.g. DAWNBench[5], KITTI Benchmark Suite[8]. These benchmarks usually focus on a couple of closely related tasks, but are not designed to host general benchmarks. In addition, they require repetitive efforts to develop and maintain, which is not always affordable by data science teams. Thus, to facilitate benchmarking, we need a platform to allow users to **flexibly and easily** create benchmarks with **custom evaluation protocols and custom data formats**, and **execution in a controlled reproducible environment**, which is totally **free and open-sourced**.

To answer these unmet needs, we developed *Codabench*, a **meta-benchmark** platform (Figure 1). A meta-benchmark platform is designed to support general purpose benchmarks and to facilitate the organization and usage of benchmarks. *Codabench* takes into account three types of contributors: benchmark participants, benchmark organizers and platform developers. Benchmark participants submit to different benchmarks, which are prepared and owned by different benchmark organizers. **Reproducibility** is required at this stage for fair benchmarking. Platform developers contribute different features to *Codabench* to support diverse benchmarks instead of one specific benchmark, i.e. *Codabench* is at the meta level of benchmarks. **Flexibility and easiness** to organize and use benchmarks are thus required at this stage. *Codabench* realizes these features by implementing an ingestion/scoring programming paradigm, supporting multiple benchmark creation methods and API access, and using Docker to guarantee reproducibility. *Codabench* has received over 130 users and 2500 submission on 100 tasks including Automated Machine Learning (AutoML)[11], Graph Machine Learning[10], Reinforcement Learning (RL)[18], detecting cancer heterogeneity and training clinicians[7]. Multiple illustrative use cases are introduced. *Codabench* is an important step towards reproducible research and should meet the interest of all areas of data science.

[2] https://www.kaggle.com/
[3] https://tianchi.aliyun.com/
[4] https://codalab.lisn.upsaclay.fr/
[5] https://archive.ics.uci.edu/ml
[6] https://paperswithcode.com/
[7] https://cancer-heterogeneity.github.io/

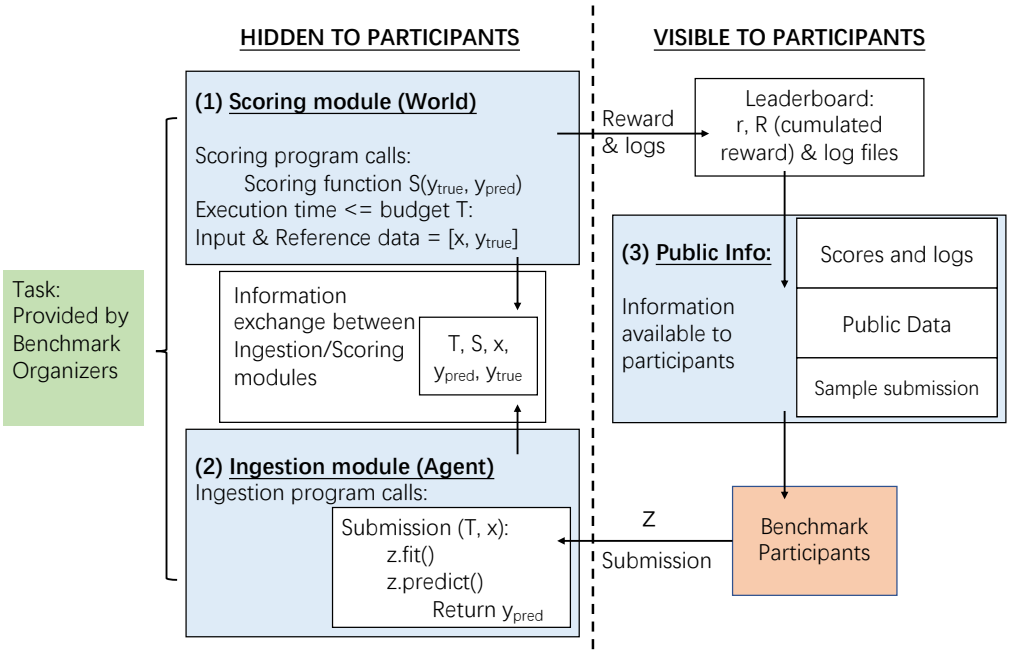


Figure 2: **Operational Codabench workflow.** Left side: Task module specified by the benchmark organizers, executed on the platform. Right: Web interface with participants permitting to make submissions and retrieve results. Numerated blocks are specified by the benchmark organizers. They include (1) a scoring module; (2) an ingestion module; (3) and public information. An intermediate block also exists for information exchange of time budget, scoring, input data, ground truth data and predictions. Red bottom right block: participant prepares a submission "z" uploaded to the platform. The submission is then executed by the ingestion program. The role of the scoring program is to produce scores that are then displayed on the leaderboard.

2 Method: Design of Codabench

Codabench is a meta-benchmark platform to provide flexible, easy-to-use and reproducible benchmarking service that is publicly and freely available for everyone. In *Codabench*, benchmarks are implemented by benchmark `bundles` which contain one or several tasks. The concept of a `task` is newly introduced, which is the minimal unit for composing a benchmark (`bundle`). A `task` consists of an "ingestion module" (including an ingestion program and input data), a "scoring module" (including a scoring program and reference data, invisible to the participant's submission), a baseline solution with sample data, and meta-data information if needed. Tasks in *Codabench* may be programmed in any programming language in any custom way, which are run in a docker specified by organizers. Figure 2 provides a detailed description of *Codabench* internal interaction logistics.

Take supervised learning tasks as an example. A typical usage is that benchmark participants submit a class (e.g. a Python object) "z", with 2 methods: `z.fit` and `z.predict`, similarly to scikit-learn objects [15]. The ingestion program reads data, calls `z.fit` with labeled training data and `z.predict` with unlabeled test data (labeled training data and unlabeled test data being part of the so-called "input data"), then outputs predictions. The scoring program reads the predictions and evaluates them based on custom scoring metric(s), using the test labels (called "reference data"). Other application usages are possible, including: transposed benchmarks, where datasets are submitted by participants instead of algorithms while the organizers supply a set of algorithms, and reinforcement-learning benchmarks, where the ingestion program plays the role of an agent wrapping around the submission of the participant and interacting with a world (scoring program) in a specific way.

The reader is referred to Codabench official repository[8], where the code and complete documentation are found. In the supplementary, we also include instructions and references to get started. To use the public instance of *Codabench*, please visit the Codabench website. To test and install locally, the instructions are given in the README file of the official repository. The *Codabench* code is released under an Apache 2.0 License.

[8] `https://github.com/codalab/codabench/`

Table 1: **Comparison of various reproducible science platforms.** Different features are introduced in Sec 3.1. 'Bundle' means whether a wrap up is provided for a benchmark such that we could reuse or share. 'Result/Code/Dataset' submit means whether different submissions are supported to enable flexible tasks. 'Compute queue' means, where public or private computation resources could be provided or linked for convenient deployment.

Platform	Flexibility			Easy-to-use				Reproducibility
	Bundle	Result/Code submit	Dataset submit	Easy creation	Open-source/free	API access	Compute queue	
Kaggle	✘	✔	✘	✔	✘	✔	✔	✔
Tianchi	✘	✔	✘	✔	✘	✘	✔	✔
CodaLab	✔	✔	✘	✔	✔	✘	✔	✔
UCI	✘	✘	✔	✘	✔	✘	✘	✔
OpenML	✘	✔	✔	✔	✔	✔	✘	✔
PapersWithCode	✘	✔	✘	✔	✔	✘	✘	✔
DAWNBench	✘	✔	✘	✘	✔	✘	✘	✔
Codabench	✔	✔	✔	✔	✔	✔	✔	✔

3 Results

3.1 Key features of Codabench

Codabench is *task-oriented*. Using tasks, the organizers have the flexibility of implementing any benchmark protocol, with any dataset format and API, or even using data generating models, allowing them to organize reinforcement learning challenges. In this section, we introduce the key features of *Codabench* contributing to the flexibility, easiness and reproducibility, as shown in Figure 1. *Codabench* also supports custom leaderboards and has full documentation of usage.

3.1.1 Flexibility

Codabench supports flexible benchmark types including results submission, code submission and even dataset submission. Benchmarks on *Codabench* are organized by bundles containing all the information of a benchmark.

Bundle (hosting a benchmark). A benchmark bundle is a zip file containing all necessary constituents of a benchmark: tasks, documentation, and configuration settings (such as leaderboard settings). A *Codabench* bundle may include single or multiple tasks. Classical benchmark is usually single-task while AutoML [11], Transfer Learning[14], Meta Learning[20] benchmarks are multi-task.

Results or code submission. "Classic" *Codabench* benchmarks are either with result or code submission. On one hand, result submissions are used when organizers wish that participants use they own computational resources. In supervised learning competitions, participants would supply e.g. predictions of output values on some test datasets. Other types of results may be supplied, for instance high resolution images in a hyper-resolution challenge for which inputs are low-resolution images. On the other hand, if the organizers wish to run all algorithms in a uniform manner on the platform, *Codabench* allows the participants to make code submissions. The submitted software is run in a docker supplied by the organizers, either on the default compute worker, or on compute workers supplied by the organizers. This code submission design allows organizers to provide suitable computational resources (e.g. GPUs), and improve reproducibility.

Dataset submission. The role of dataset and algorithm can be transposed with *Codabench* to facilitate **Data-Centric AI**[9] which is a trending research topic that cares about the quality and usage of data. In a "classic" benchmark, organizers provide datasets and participants submit algorithms. In a transposed benchmark, participants submit datasets and organizers provide reference algorithms. A "classic" benchmark will have a leaderboard with datasets in columns, growing by adding more lines as algorithm submissions are made. In a transposed dataset submission benchmark, the leaderboard will have algorithms in columns and lines are added as more datasets are submitted. With Codabench's transposed benchmark, it is easier to try different data augmentation and processing methods with fixed algorithms as test cases.

[9] `https://datacentricai.org/`

3.1.2 Easy-to-use

To facilitate an easiness of using the system, we provided several tools to help the benchmark organizers to create a benchmark. A platform editor is provided to develop a benchmark, which provides simple user-interfaces to prepare data, code, and other configurations. As a second option, the user can upload a locally prepared benchmark bundle to facilitate local debugging and testing. Once uploaded, a benchmark can further be modified using the platform editor. An existing benchmark can be saved as another bundle, which facilitates the sharing and portability. Similar benchmark bundles can be easily prepared with shared template bundles. *Codabench* is **open-sourced and free** to use.

APIs to external clients. We provide APIs for interacting with the platform, including "robot" submissions via command lines, without going through the regular *Codabench* web interface, and likewise a programmatic way of recuperating results directly without going through the leaderboard.

Dedicated computing queues. The public instance of *Codabench* provides default compute workers. Organizers can also create a dedicated job queue and connect it to their own CPU or GPU compute workers.

3.1.3 Reproducibility

Codabench makes extensive use of Dockers[10] to guarantee reproducibility. Benchmark organizers specify the Docker image by providing its Docker Hub name and tag. Docker wraps all the software dependencies into a lightweight virtual image. Once a docker is provided by the benchmark organizer, the program can be run inside a docker which contains exactly the same installed packages. This docker will be pulled every time a benchmark's program is executed. Different benchmarks could use different dockers, which are usually provided by organizers. We also provide a default docker for more general benchmarks' usage or people who are not familiar with dockers.

3.1.4 Other features

Custom leaderboard. To better facilitate benchmarks, the leaderboard is fully customizable and can handle multiple datasets and multiple custom scoring functions. We provide multiple ways to display submissions (best per participant, multiple submissions per participant, etc.) and the leaderboard can flexibly rank submissions by average score, per task, per sub-metric of a certain task, etc.

Documentation. The documentation[11] provides detailed help for different types of contributors. For benchmark participants, we provide instructions to join and submit to a benchmark. For benchmark organizers, we provide annotated instructions for organizing benchmarks. Several benchmark bundle templates, from simple to advanced, are also available to ease the technical aspects of building a benchmark, and to let people concentrate on scientific aspects of the benchmark. For platform developers, we explain more technical specifications on technology stack and provide ways to integrate to the project. Platform developers are contacted via the GitHub issues and pull requests to solve issues encountered in daily usage.

3.2 Use cases of Codabench

Codabench has been used not only internally at 4Paradigm and LISN Lab for tasks of AutoML[11], Graph Machine Learning[10], Reinforcement Learning[18], Speech Recognition[16] and Weakly Supervised Learning[21], but also externally by University Grenoble Alphes for hosting scientific benchmarks in cancer heterogeneity and training clinicians. *Codabench* has received more than 130 users and 2500 submissions distributing on various applications. In this section, we introduce 4 use cases of *Codabench*, aiming at demonstrating different *Codabench* features and capabilities. A visual illustration is given in Figure 3.

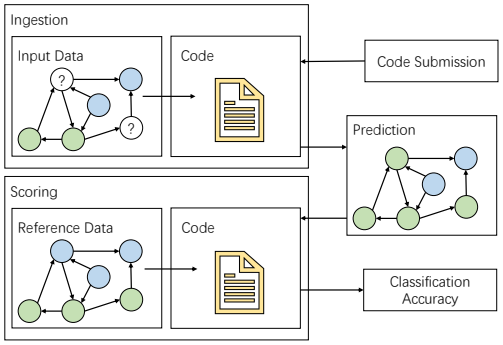
3.2.1 Use case 1: AutoGraph benchmark

In this section, we introduce Automated Graph Machine Learning (AutoGraph) benchmark, which targets at automated node classification methods on diverse dataset scenarios. With this use case, we show a set of fundamental features of *Codabench*: **(1) the code submission mode (2) reproducibility guaranteed by docker (3) flexible benchmark bundle configuration with multiple tasks, and (4) customizable computational resources.**

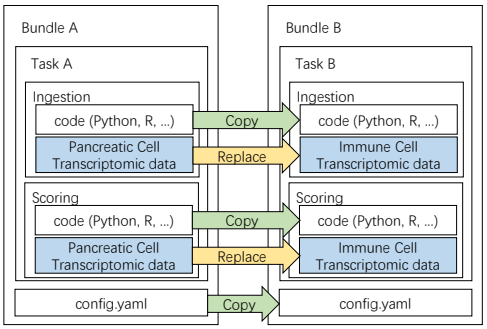
Background. Graph Machine Learning has been a very hot topic due to ubiquity of graph-structured data, e.g. social network[9], molecule graph[13], knowledge graph[3], etc. Typical tasks of graph data include node-level (node classification), edge-level (link prediction) and graph-level (graph regression/classification). The task of our benchmark here is node

[10] `https://www.docker.com/`

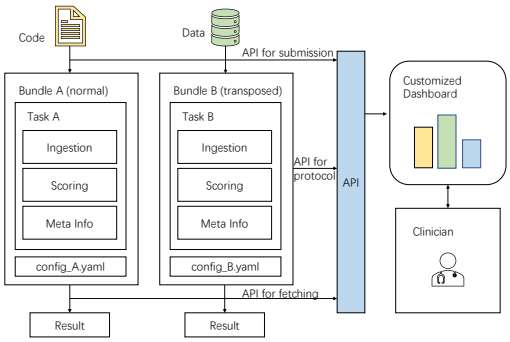
[11] `https://github.com/codalab/codabench/wiki`



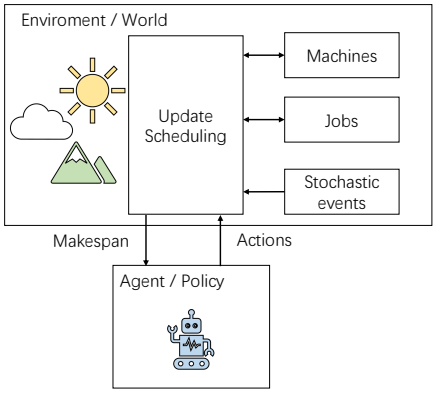
(a) Use case 1: AutoGraph



(b) Use case 2: DECONbench



(c) Use case 3: COMETH



(d) Use case 4: Job Scheduling

Figure 3: **Use case illustrations.** The details are introduced in Section 3.2.

classification, i.e. given a graph where some nodes are labeled and the rest are unlabeled, we want to predict the classes of the unlabeled nodes. In addition, we require the algorithm to perform well on a set of datasets instead of just one dataset. This leads to Automated Graph Machine Learning problem which we call AutoGraph.

Implementation. The AutoGraph benchmark is a typical **code submission** use case. It focuses on Automated Machine Learning (AutoML) methods [11] which requires more than one dataset to be evaluated together. *Codabench* bundle is by design flexible with **multiple tasks** each of which contains a seperate dataset. We also provide a docker hosted on DockerHub, which could be pulled automatically by *Codabench* platform to run each algorithm submission and be used for researchers' local development. Every time a new method is uploaded, a new docker container instance will be called to independently run the method for each dataset. In this way, we make sure every algorithm is fairly run under the same setting and the whole process can be **fully reproduced** on other machines. *Codabench* is designed to adapt to any Docker-enabled computational resource (local machine, cluster server, cloud machines, etc.). We currently host the AutoGraph benchmark on *Codabench* with **free computational resources** thanks to Google's sponsorship, encouraging everyone to contribute. Besides, the datasets are also available to the public for local usage and further benchmarking on GitHub[12] and Kaggle. We uploaded the solutions of the winners of the challenge as baselines. Since the benchmark datasets are already released, users can also run complementary experiments on their local computers and debug mode easily, thus more rapidly making progress. With AutoGraph benchmark, we provide researchers and practitioners the possibility to showcase results in a public venue.

3.2.2 Use case 2: DECONbench benchmark

In this section, we introduce the DECONbench benchmark [7]. DECONbench aims at benchmarking algorithms inferring tumor cellular composition from molecular data. Here we highlight two features of *Codabench*: **(1) flexibility of benchmark bundle (in this use case, another task and programming language R supported) (2) reusability and portability of benchmark bundles**

[12] `https://github.com/AutoML-Research/AutoGraph-KDDCup2020`

Background. Successful treatment of cancer is still a challenge and this is partly due to a wide heterogeneity of tumor cellular composition across patient population. Tumors are made up of cells with different identities and origins. Cancer cells evolving in a dynamic environment consist of aberrant non-cancerous cells, such as blood vessels or immune cells. Tumor cellular composition is difficult to observe and quantify, as it is hidden inside the bulk molecular profiles of the samples, with millions of cells present in the tumor (and not only cancer cells) contributing to the bulk recorded signals. Taking advantage of the large amount of molecular data publicly available, a wide number of supervised and unsupervised algorithms have been recently developed to estimate tumor cellular composition[4;1;6]. DECONbench is a **series of benchmarks** dedicated to the quantification of tumor composition, focusing on estimating cell-types and proportion in biological samples using multimodal molecular data. Participants have to identify an estimation of the tumor composition, i.e. a matrix of estimated proportion of each deconvoluted cell-type (rows) for each sample (columns). The discriminating metric is mean absolute error (MAE) between prediction and ground truth matrix. Note that DECONbench series is optimized to run methods developed in the statistical **programming language R**.

Implementation. Using the Codabench platform, the COMETH consortium firstly developed a benchmark for continuous evaluation of computational methods based on epigenomic data[13]. Since we are at the same time interested in other modalities of data under similar tasks, it would be ideal to reuse previously created bundles instead of going through everything again. Thanks to the portability of *Codabench* bundle design, we only need to replace the data files and adjust slightly the protocol code. All other configuration files can be reused. As a result, this first benchmark was easily cloned and extended to similar benchmarks using other types of data, e.g. all-cell-type transcriptomic data[14], immune-cell-types transcriptomic data[15], all-cell-types multimodal transcriptomic and epigenomic data[16].

3.2.3 Use case 3: COMETH benchmark

In this section, we introduce the COMETH benchmark, motivated by real clinical application and it is an exciting step towards Data-Centric AI. With this use case, we show that **(1) *Codabench* supports a transposed benchmark consolidating Data-Centric AI (2) the provided API interaction opens a window for other customization scenarios.**

Background. When it comes to clinical application, it is often necessary for health data scientists and clinicians to identify the most suitable existing method to be applied on a given dataset. In this case, we focus more on the data instead of algorithmic development, which aligns with Data-Centric AI. Usually, the clinicians do not (need to) know much about the algorithm details. Instead, they have access to newly available data and want to apply the most relevant algorithms on their new data. There is thus a need to provide an effective tool displaying the evaluation of state-of-the-art algorithms on reference dataset, and enabling their application on new dataset. This will guide and facilitate the appropriate use of these algorithms by non-expert clinicians. Note that these algorithms are usually provided by benchmark organizers who are domain experts on certain tasks.

Implementation. To solve this question, the COMETH consortium developed the COMETH benchmark [17], a transposed challenge in which datasets should be submitted to be evaluated against existing algorithms (i.e. `tasks` in the Codabench design). For instance, the COMETH benchmark provides a series of recent deconvolution algorithms that are able to quantify tumor heterogeneity[4;1;6]. Clinicians aiming to quantify tumor heterogeneity from molecular data can submit their dataset of interest to the COMETH benchmark and retrieve the corresponding outputs, in a fully reproducible environment. To facilitate the use of this functionality by clinicians who are less familiar with data science programming details, COMETH benchmark has been connected to an external client displaying a user-friendly web dashboard. The external client is able to send requests to users directly on the COMETH benchmark using APIs provided by *Codabench* and return the generated results from all reference algorithms. This feature strongly contributes to a direct transfer of knowledge between data scientists and healthcare professionals. This design was used at a winter school for training clinicians and data scientists [18].

3.2.4 Use case 4: Job Scheduling benchmark

We lastly introduce another use case: Job Scheduling benchmark focusing on reinforcement learning and operational research. With this use case, we show that *Codabench* is **RL-friendly** with the help of flexible design of benchmark bundles.

[13] `https://www.codabench.org/competitions/174`
[14] `https://www.codabench.org/competitions/147`
[15] `https://www.codabench.org/competitions/148`
[16] `https://www.codabench.org/competitions/237`
[17] `https://www.codabench.org/competitions/218`
[18] `https://cancer-heterogeneity.github.io/cometh.html`

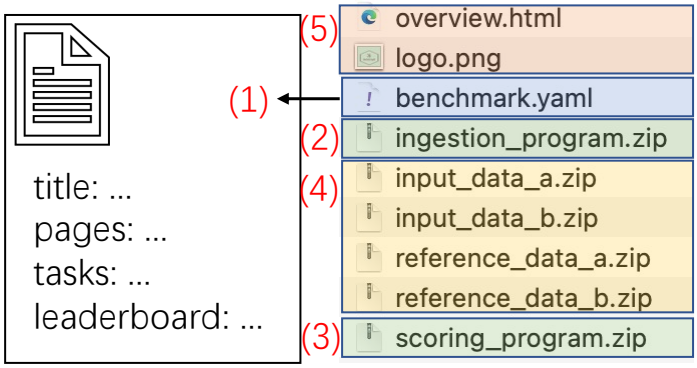


Figure 4: **Bundle structure.** The details of `benchmark.yaml` is given in Data S1.

Background. We consider the problem of Dynamic Job Shop Scheduling[2;17;12]. The task is to allocate a set of jobs to a set of machines to achieve the shortest execution time, i.e. makespan. Each job has a pre-determined operation sequence to be executed on certain machines. To mimic real life scenarios, we add stochastic machine down events to the problem. This task is usually formulated as a sequential decision-making problem and fits easily to reinforcement learning. We thus expect an agent making decisions on how to schedule better the jobs in minimal time. The reward depends on the makespan.

Implementation. As explained in Sec 2, our design of bundle and ingestion/scoring program makes it very natural and flexible for RL problems. We easily use the scoring program as an environment which evaluates a job schedule and returns a makespan as reward. The ingestion program serves as an agent and makes decisions on job scheduling based on received reward.

3.3 A concrete benchmark bundle example

In this section, we provide a concrete benchmark bundle example to show how simple it is to organize benchmarks on *Codabench*. A bundle consists of five parts as in Figure 4: (1) a YAML configuration file[19], (2) ingestion program, (3) scoring program, (4) data and (5) additional files for description.

The **ingestion program** usually reads data and participant's submission. It calls participant's method on the dataset and produces predictions to a shared space. The **scoring program** usually reads ingestion program's output and evaluate w.r.t ground truth according to organizer customized metric. It finally writes scores to a text file which will be read by the platform and be displayed on a leaderboard. The **data** contain input data (in supervised learning, they are usually X_train, y_train, and X_test) and reference data (in supervised learning, it is usually y_test). Both are zipped into separate files. The **additional files** are just text or figure files for organizers to provide other information e.g. instructions, references, logo, etc. A final **YAML file** connects all previous parts and provides more configurations for the benchmark. A simplified YAML file is given on Data S1 in the supplementary. It contains general configurations like title, logo image, docker image, and which HTMLs to be displayed, leaderboard configuration (e.g. which metrics will be used in the leaderboard) and tasks. Each task is by itself a complete unit for running. It contains name, ID, ingestion program, scoring program, input data, reference data.

4 Discussion

Codabench is a new meta-benchmark platform for data science communities. *Codabench* is compatible with diverse tasks (including supervised learning and reinforcement learning) and supports result, code, and dataset submission. It is easy to use *Codabench* and reproducibility is guaranteed by Dockers. *Codabench* has a public instance free for use, deployed at Université Paris-Saclay, but can also be deployed locally, with the technology stack provided in documentation. Hosting, maintaining, and further developing the platform is funded by grants and donations. As real-world scenarios, we introduce 4 benchmark use cases illustrating the flexibility, easiness in use, reproducibility and other features of *Codabench*. We also note that tremendous other tasks could be integrated into *Codabench* as well

[19] `https://yaml.org/`

including EEG classification, drug discovery and property prediction, dynamic simulation for weather, traffic, fluid, etc., which are important tasks towards AI for Science.

The current limitations of *Codabench* are mainly as follows. First, since it is relatively new, we do not have yet an active community of organizers and benchmark participants. We need more users' feedbacks to polish up our user-interface and documentation. Second, although supported by design, we have not had yet a distributed computation scenario, where complex multi-node compute workers are used. This could enrich our benchmark template library with benchmarks for algorithm parallelization. Thirdly, although *Codabench* supports both code submission and dataset submission, we do not currently allow users to extend the leaderboard in both directions simultaneously, i.e., it does not allow users to submit both code and datasets at the same time. This feature could largely increase the user experience of the platform. Lastly, *Codabench* doesn't support yet hardware related benchmarks or human-in-the-loop benchmarks which could be interesting to consider in the future.

Potentially harmful uses of *Codabench* could result from poor benchmark designs (e.g., no scientific question is asked by hosting a benchmark), or bad data collections (e.g., data license, data quality), as in any machine learning project. We are working on an open-access book (to appear in 2022) on best practices for designing challenges and benchmarks including data preparation, task evaluation, benchmark analyse paper, etc.

Further works include providing more comprehensive usage templates illustrating features such as: (1) splitting an algorithm workflow into submodules and scoring the effectiveness of the modules individually (e.g., with ablation or sensitivity analysis); (2) providing templates of fact sheets to extract information about algorithms (similar to datasheets for datasets, but for algorithms); and (3) providing guidelines to benchmark participants to produce enriched detailed results, amenable to meta-analyses.

5 Experimental Procedures

5.1 Resource availability

Lead Contact. Zhen Xu (`xuzhen@4paradigm.com`). 4Paradigm, Beijing 100085, China.

Materials availability. This study did not generate new materials.

Data and code availability. The code of Codabench is available at `https://github.com/codalab/codabench`. This work does not introduce new datasets. For creating benchmarks, organizers should prepare their own datasets.

Acknowledgments

The Codabench project shares the same community governance as *CodaLab Competitions*. The openness of *Codabench* is total: the Apache 2.0 licence is used, the source code is on GitHub; the development framework and all the used components are open-sourced. *Codabench* has received important contributions from many people who did not co-author this paper, and we would like to thank their efforts in making *Codabench* what it is today, including early *CodaLab Competitions* developers and contributors (alphabetically): Pujun Bhatnagar, Justin Carden, Richard Caruana, Francis Cleary, Xiawei Guo, Ivan Judson, Lori Ada Kilty, Shaunak Kishore, Stephen Koo, Percy Liang, Zhengying Liu, Pragnya Maduskar, Simon Mercer, Arthur Pesah, Christophe Poulain, Lukasz Romaszko, Laurent Senta, Lisheng Sun, Sebastien Treguer Cedric Vachaudez, Evelyne Viegas, Paul Viola, Erick Watson, Tony Yang, Flavio Zhingri, Michael Zyskowski. We would like thank particularly the people who contributed to the design, development, and testing of *Codabench* including (alphabetically): Alexis Arnaud, Xavier Baró, Feng Bin, Yuna Blum, Eric Carmichael, Laurent Darré. Hugo Jair Escalante, Sergio Escalera, Eric Frichot, Yuxuan He, James Keith, Anne-Catherine Letournel, Shouxiang Liu, Zhenwu Liu, Adrien Pavao, Magali Richard, Tyler Thomas, Nic Threfts, Bailey Trefts, Catherine Wallez, Lanning Wei. Université Paris-Saclay is hosting the main instance of *Codabench*. Funding and support have been received by several research grants, including Big Data Chair of Excellence FDS Paris-Saclay, Paris Région Ile-de-France, EU EIT projects HADACA and COMETH, United Health Foundation INCITE project, ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022, the Spanish project PID2019-105093GB-I00, ICREA under the ICREA Academia programme, Inserm Cancer project ACACIA 232717, MIAI @Grenoble Alpes (ANR-19-P3IA-0003), 4Paradigm, ChaLearn, Microsoft, Google. We also appreciate the following people and institutes for open sourcing datasets which are used in our use cases: Andrew McCallum, C. Lee Giles, Ken Lang, Tom Mitchell, William L. Hamilton, Maximilian Mumme, Oleksandr Shchur, David D. Lewis, William Hersh, Just Research and Carnegie Mellon University, NEC Research Institute, Carnegie Mellon University, Stanford University, Technical University of Munich, AT&T Labs, Oregon Health Sciences University. We are also very grateful to Joaquin Vanschoren for fruitful discussions.

Author Contributions

Conceptualization, Z.X, S.E, A.P, I.G ; Methodology, Z.X, I.G; Validation & Investigation, all authors; Resources & Data Curation, Z.X, M.R, W.W.T, I.G; Writing – Original Draft, all authors; Writing – Review & Editing, Z.X, Q.Y, M.R, I.G; Visualization, Z.X, Q.Y, I.G; Supervision & Project Administration, I.G; Funding Acquisition, W.W.T, I.G.

Declaration of Interests

Author Z.X, WW.T and H.Z are employed by 4Paradigm, China. Author IG is president of ChaLearn, a non-for-profit organization dedicated to running challenges in machine learning. ChaLearn is a tax exempt non-for-profit organization under section 501(c)(3) of the US IRS code of the Unites States. It derived no profit from sponsoring this research. All authors declare no other competing interests.

References

[1] Francisco Avila Cobos, José Alquicira-Hernandez, Joseph E Powell, Pieter Mestdagh, and Katleen De Preter. Benchmarking of cell type deconvolution pipelines for transcriptomics data. *Nature communications*, 11(1):1–14, 2020.

[2] M Emin Aydin and Ercan Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 2000.

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems*, 26, 2013.

[4] Laura Cantini, Ulykbek Kairov, Aurélien De Reynies, Emmanuel Barillot, François Radvanyi, and Andrei Zinovyev. Assessing reproducibility of matrix factorization methods in independent transcriptomes. *Bioinformatics*, 35(21):4307–4313, 2019.

[5] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Operating Systems Review*, 53(1):14–25, 2019.

[6] Clémentine Decamps, Florian Privé, Raphael Bacher, Daniel Jost, Arthur Waguet, Eugene Andres Houseman, Eugene Lurie, Pavlo Lutsik, Aleksandar Milosavljevic, Michael Scherer, Michael G. B. Blum, and Magali Richard. Guidelines for cell-type heterogeneity quantification based on a comparative analysis of reference-free dna methylation deconvolution software. *BMC bioinformatics*, 21(1):1–15, 2020.

[7] Clémentine Decamps, Alexis Arnaud, Florent Petitprez, Mira Ayadi, Aurélia Baurès, Lucile Armenoult, Sergio Escalera, Isabelle Guyon, Rémy Nicolle, Richard Tomasini, et al. Deconbench: a benchmarking platform dedicated to deconvolution methods for tumor heterogeneity quantification. *BMC bioinformatics*, 22(1):1–17, 2021.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE Computer Society, 2012.

[9] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.

[10] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, 2020.

[11] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[12] Anant Singh Jain and Sheik Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 1999.

[13] Elman Mansimov, Omar Mahmood, Seokho Kang, and Kyunghyun Cho. Molecular geometry prediction using a deep generative graph neural network. *Scientific reports*, 2019.

[14] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 2009.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 2011.

[16] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE workshop on automatic speech recognition and understanding*, 2011.

[17] R Ramasesh. Dynamic job shop scheduling: a survey of simulation research. *Omega*, 1990.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[19] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[20] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[21] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National science review*, 2018.