Codabench: Flexible, Easy-to-Use and Reproducible Benchmarking for Everyone

Zhen Xu, Huan Zhao, Wei-Wei Tu, Magali Richard, Sergio Escalera, Isabelle Guyon

Codabench: Flexible, Easy-to-Use and Reproducible Benchmarking for Everyone

Zhen Xu[1], **Huan Zhao**[1], **Wei-Wei Tu**[1,5], **Magali Richard**[2], **Sergio Escalera**[3], **Isabelle Guyon**[4,5]
1 4Paradigm, Beijing, China
2 TIMC-IMAG, UMR5525, Univ. Grenoble Alpes, CNRS, Grenoble, France
3 Universitat de Barcelona and Computer Vision Center, Spain
4 LISN (CNRS/INRIA) Université Paris-Saclay, France
5 ChaLearn, California, USA
xuzhen@4paradigm.com

Abstract

Obtaining standardized crowdsourced benchmark of computational methods is a major issue in scientific communities. Dedicated frameworks enabling fair continuous benchmarking in a unified environment are yet to be developed. Here we introduce Codabench, an open-sourced, community-driven platform for benchmarking algorithms or software agents versus datasets or tasks. A public instance of Codabench 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 the organization of benchmarks flexibly, easily and reproducibly. Firstly, it supports code submission and data submission for testing on dedicated compute workers, which can be supplied by the benchmark organizers. This makes the system scalable, at low cost for the platform providers. Secondly, Codabench benchmarks are created from self-contained "bundles", which are zip files containing a full description of the benchmark in a configuration file (following a well-defined schema), documentation pages, data, ingestion and scoring programs, making benchmarks reusable and portable. The Codabench documentation includes many examples of bundles that can serve as templates. Thirdly, Codabench uses dockers for each task's running environment to make results reproducible. Codabench has been used internally and externally with more than 10 applications during the past 6 months. As illustrative use cases, we introduce 4 diverse benchmarks covering Graph Machine Learning, Cancer Heterogeneity, Clinical Diagnosis and Reinforcement Learning.

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 fair evaluation, with which metrics, under which software/hardware configurations, and on which platform. The concept of benchmark itself is not well standardized and includes many different 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 algorithm benchmarks do not offer the easiness to further integrate new methodological developments. A platform for benchmarking tasks in a flexible and reproducible way is thus much needed for everyone to use.

Preprint. Under review.

Table 1: Comparison of various reproducible science platforms.☆ means that this feature is not or minimally supported. ☆☆ means that some efforts have been put to support this feature. ☆☆☆ means that this feature is well supported. "Reproducibility" means whether we can easily reproduce the reported performance. "Portability" means whether a certain benchmark design makes creating another one easier. "Data-Centric" means whether the platform has a focus on data, e.g. hosting datasets, submiting datasets instead of methods. "API access" means whether we could interact with the platform through command line and eventually develop customized applications. "RL-friendly" means whether the platform supports this important task by design. "Computation resource" means whether machine resources are provided or easily managed. "Open Source" means whether we could deploy our own version of the platform through public materials. "Free usage" means whether we are free to organize benchmarks or submit solutions.

	Reproducibility	Portability	Data-Centric	API access	RL-friendly	Computation resource	Open Source	Free usage
Kaggle	☆☆	☆☆	☆☆	☆☆	☆☆☆	☆☆☆	☆	☆☆
Tianchi	☆☆	☆☆	☆☆	☆	☆☆☆	☆☆	☆	☆☆
UCI	☆	☆	☆☆	☆	☆	☆	☆☆☆	☆☆☆
OpenML	☆☆	☆☆	☆☆	☆☆	☆☆	☆	☆☆☆	☆☆☆
PapersWithCode	☆	☆☆	☆	☆	☆☆	☆	☆	☆☆☆
DAWNBench	☆	☆	☆	☆	☆	☆	☆	☆☆☆
CodaLab	☆☆☆	☆☆☆	☆	☆	☆☆	☆☆	☆☆☆	☆☆☆
Codabench	☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆	☆☆☆	☆☆☆

Typical examples of existing frameworks addressing such need are inventoried in Table 1. Firstly, they include competition platforms, such as Kaggle and Tianchi organizing many data science challenges attracting a large number of participants. They provide elaborate ways of hosting third party competitions and offer services for a fee for commercial competitions. The platform providers retain some control: the organizers do not have full flexibility and control over their competitions. Secondly, data repositories such as UCI repository[5] also play an important role for benchmarks and research. But they do not host methods, or results. In contrast, OpenML[22] is an example of open-sourced and free hub of datasets also making available machine learning results. However, reproducibility by running code in given containers (or similar ways) is not guaranteed. Similarly, PapersWithCode collects many tasks and state of the art results from papers. But the platform doesn't guarantee the reproducibility of these performances. Besides the above mentioned platforms, many domain specific benchmarks exist, e.g. DAWNBench [3], KITTI Benchmark Suite[7]. 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. **We thus need a platform DEMOCRATIZED FOR EVERYONE that supports diverse benchmark types, facilitates benchmark organization, and guarantees reproducibility.**

To answer these unmet needs for benchmark platforms, we developed *Codabench* to allow users to **flexibly and easily** create benchmarks with **well-defined custom evaluation protocols and custom data formats,** and **execution in a controlled reproducible environment**, which is totally **free and open sourced**. *Codabench* is an important step towards reproducible research and should meet the interest of all areas of data sciences.

Codabench is the last born of a suite of tools from the open-source"ChaSuite" (Figure 1), which all have public instances available for use free of charge. "ChaSuite" provides a comprehensive suite of tools for competition and benchmark organizers. *Codabench* is inspired by *CodaLab Competitions*, an open source platform for running data science competitions, which has been used in hundreds of challenges associated to physics, machine learning, computer vision, natural language processing, health and life sciences, among many other fields. Data science competitions have played an important role for solving machine learning problems both in theory and application (e.g. ImageNet challenge [20], the Netflix Prize [1], the 1714 Longitude Prize [19], etc). Benchmarks can be view as a never-ending competition enabling continuous evaluation of methods under the same settings (see Table 2 for a comparison between benchmark and competitions).

Compared with *CodaLab Competitions*, *Codabench* has made significant improvements to better address the organization of benchmarks. The full code has been completely rewritten and the code

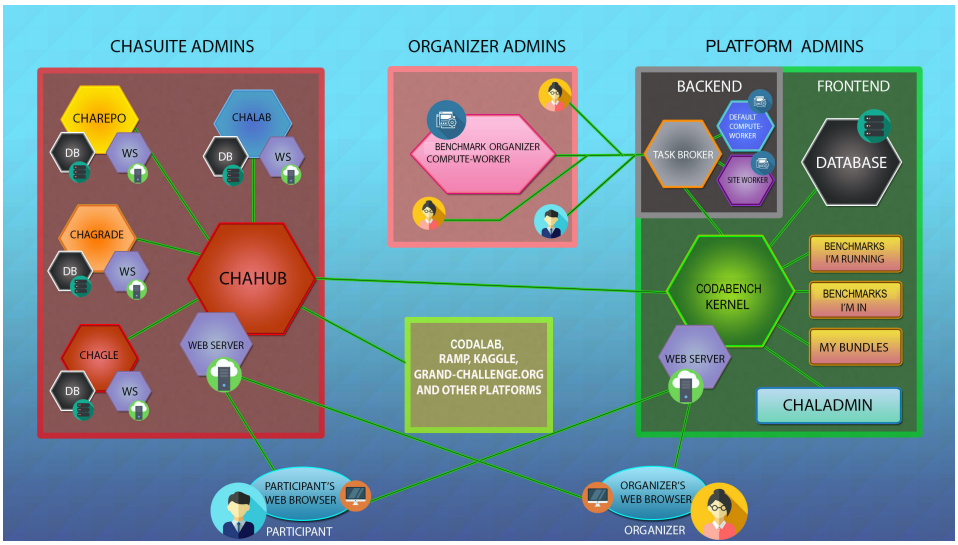


Figure 1: **ChaSuite architecture.** *Codabench* is part of the *CodaLab Competitions* project, including a suite of tools to organize challenges and benchmarks called the"ChaSuite". Right: The kernel of Codabench is interfaced with a web browsers, a database, and a backend dispatching jobs to compute-workers, configured and administered by organizers. Left: The ChaSuite includes an index of competitions and benchmarks (ChaHub) with a search engine (Chagle), a wizard to design challenges (ChaLab), a data repository (ChaRepo), a tool to administer classes (ChaGrade).

base is much cleaner and maintainable. We introduce a new "task" concept (as mentioned in Sec 2 and Sec 3) for flexibility and portability purposes. We now support data submission in addition to results and code submission, which makes *Codabench* an important platform for Data Centric AI[1], which is a new trending paradigm focusing more on the underlying data used to train and evaluate models. We also provide low level APIs to facilitate third party's customization. A new fact sheet system has been added to allow submit more information in an integrated way and the leaderboard now supports multiple modes of display and advanced ranking.

The remaining of this paper is organized as follows. In Sec 2, we introduce the *Codabench* platform design and explain the interaction between different modules. In Sec 3, we highlight important features of *Codabench*. In Sec 4, for illustrative purposes, we provide 4 use cases of benchmarks, each focusing on different key features of *Codabench*: Case 1 AutoGraph benchmark showing fundamental features on code submission, reproducibility, flexible benchmark design and freely available computational resources on *Codabench*; Case 2 DECONbench benchmark series showing features on portability and flexibility of benchmark bundles; Case 3 COMETH benchmark showing features on transposed benchmark and the provided APIs access; Case 4 AutoRL benchmark showing feature on easy customization for reinforcement learning task. All these use cases cover diverse tasks and application scenarios: node classification of graph-structured data, cancer heterogeneity inference, educational clinical tool and operational research of environment-agent interaction.

2 Design of Codabench

Codabench is *task-oriented* (see Figure 2 for detailed internal interaction logistics). A task (supplied by benchmark organizers) consists of an "ingestion module" (usually coupled with some "input data") and a "scoring module" (usually coupled with some "reference data", invisible to the participant's

[1] `http://datacentricai.org/`

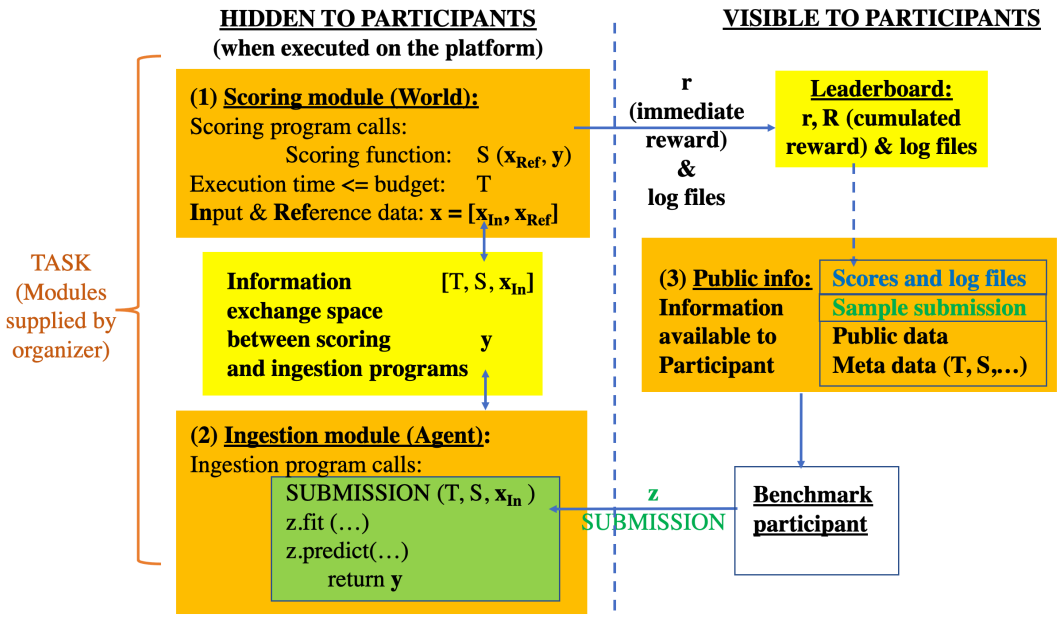


Figure 2: **Operational Codabench workflow.** Left side: Task module specified by the organizer, executed on the platform. Right: Web interface with participants permitting to make submissions and retrieve results. Orange shaded blocks are provided by organizers. They include (1) a scoring module; (2) and ingestion modules; (3) public information. White 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.

submission). Tasks may be programmed in any programming language and interfaced with data in any custom way, which are run in a docker specified by organizers.

A hallmark of Codabench is the notion of benchmark bundle, which are zip files containing the entire specification of the benchmark (including task data and code, documentation pages, and configuration parameters). Each benchmark bundle can include one or *multiple* tasks, thus covering both classic benchmarks (one dataset plus multiple algorithm submissions) as well as benchmarks on AutoML, Transfer Learning, Meta Learning (multiple datasets or multiple methods are needed). Such bundles also make it easy to export, import, archive benchmarks, and create benchmarks templates for sharing and dissemination.

Codabench is a free public platform with small computing resources accessible for basic usage. It also enables flexible and extensible computational resources supplied by the benchmark organizers: indeed, jobs submitted by a participant to a benchmark can be directed to compute-workers supplied by the organizers.

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 [18] objects. 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 (datasets are submitted by participants instead of algorithms; the organizers supply a set of algorithms), and reinforcement-learning benchmarks (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.

Table 2: Comparisons of competition and benchmark.

	Competition	Benchmark
Purpose	Crowdsourcing problems in a short time and harvesting solutions	Continuous fair evaluation, over a long time period, in a unified framework
Phases	Multiple phases	Single phase
Time period	Usually limited	Often never ending
Cooperation & information sharing	Limited due to the competitive nature	As extensive as possible
Submissions	Usually algorithm predictions or algorithm code	Algorithm code or datasets; code or dataset name, description, documentation meta-data and/or fact-sheets; scoring programs for custom analyses
Outcome	Leaderboard with usually a single global ranking based on one score from each team (last or best)	Table with all the submissions made; sorting with multiple scores possible; multiple analyses, graphs, figures, code sharing

The reader is referred to Codabench official repository[2] where the code and complete documentation are found. In Appendix, 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. Under the organization group, there is also *CodaLab Competitions*, which is the aforementioned competition platform, and CodaLab Worksheets, which features dynamical workflows, particularly useful for Natural Language Processing. This paper concerns only *Codabench*.

3 Key features of Codabench

Codabench is a flexible, easy-to-use and reproducible benchmark platform that is open sourced and freely provided for everyone. In this section, we introduce the key features of *Codabench* contributing to the flexibility, easiness and reproducibility respectively. We mention briefly other features at last.

3.1 Flexibility

Task. A new concept of task is introduced in *Codabench*. A task is composed of an "ingestion module" (including ingestion program and input data), a "scoring module" (including a scoring program and reference data), a baseline solution with sample data, and meta-data information, like name and description. It is the minimal unit for composing a benchmark. 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.

Benchmark bundle. 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 usually single-taks while AutoML, Transfer Learning, Meta Learning 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.

[2] `https://github.com/codalab/competitions-v2/`

Dataset submission. To faciliate **Data-Centric AI**, the role of dataset and algorithm can be transposed with *Codabench*. In a "classic" benchmark, organizers provide dataset(s) 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 are 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. *Codabench* does not support yet bechmarks in which both dimensions of the leaderboard are grown (*i.e.,* participants can supply either algorithms or datasets).

Dedicated queues and compute workers to add external computational resources). The public instance of *Codabench* provides default compute workers. However, to run computationally demanding benchmarks, organizers can create a dedicated job queue and connect it to they own CPU or GPU compute workers (physical or virtual machines on any cloud service), which listen to the queue dispatching jobs and pick them up on a first-come first serve basis. This modular architecture of *Codabench* is a key ingredient to grow its usage flexibly, without requiring that the institution hosting the main instance covers all computational costs. Another interesting aspect of this feature is that the training and testing of algorithms can be done on confidential data, without any leakage, by putting the datasets directly inside the compute workers. This should be useful in particular for medical research, industry benchmarks, and other restricted domains.

3.2 Easy-to-use

Multiple benchmark creation methods. A benchmark can be created either with platform editor or by uploading a locally prepared benchmark bundle. Once created, 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. There is also an option to download a light version of the bundle, without datasets and/or programs; the configuration file then points to database endpoints on the platform. This facilitates sharing and cloning benchmarks without having to re-upload data or code.

APIs to external clients. We provide APIs[3] 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.

3.3 Reproducibility

Benchmark environments using Docker image. *Codabench* makes extensive use of Dockers. Benchmark organizers specify the Docker image by providing its Docker Hub name and tag. The ingestion program and the scoring program are run separately in different Docker containers. This ensures that "reference data" is inaccessible to the participant code (if relevant). All algorithms are evaluated in the same way, and the benchmark does not get deprecated after some time by inadequate library updates. It is with Docker that *Codabench* provides full **reproducibility** to everyone.

3.4 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 ranking submissions by average score, per task, per sub-metric of a certain task, etc.)

Documentation. The documentation is organized according to stakeholders categories *organizers*, *administrators*, and *contributors* directly on the first page of the documentation [4]. As an *organizer*, you are accompanied with several benchmark templates, from simple to elaborate, to ease the technical aspects of building a benchmark, and to let you concentrate on scientific aspects of the benchmark. As an *administrator* of your own instance of *Codabench*, each piece of the infrastructure is configurable and offered as a docker component. You can deploy your instance in a very flexible way concerning the sizing of your project thanks to deployment guide hints. As an *contributor*, you

[3] `https://github.com/codalab/competitions-v2/wiki/Robot-submissions`

[4] `https://github.com/codalab/competitions-v2/wiki`

can discuss with the main developers via the GitHub issues and suggest pull requests to solve some of the issues you have encountered.

Backward compatibility with Codalab. While *Codabench*'s novelty is the possibility of creating benchmarks, it is fully compatible with *CodaLab Competitions*. Competition bundles in the old format *e.g.,* dumped from the Codalab public instance can be re-uploaded to *Codabench*. Competition features such as having multiple-phases (not usually relevance for benchmarks) are supported for compatibility reasons in *Codabench*. Multi-phase challenges help organizers keep participants engaged over long periods of time.

4 Use cases of Codabench

Codabench has been used not only internally at 4Paradigm and LISN Lab for tasks of AutoML, Graph Machine Learning, Reinforcement Learning, Speech Recognition and Weakly Supervised Learning, but also externally by University Grenoble Alphes for hosting scientific benchmark in cancer heterogeneity and training clinicians. A total of more than 10 use cases are developed during the past 6 months. In this section, we introduce 4 use cases of *Codabench*, aiming at demonstrating different *Codabench* features and capabilities.

4.1 Use case 1: AutoGraph benchmark

In this section, we introduce Automated Graph Representation 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.** More technical details can be found in Appendix.

Background. The AutoGraph benchmark inherits from the Automated Graph Representation Learning (AutoGraph) Challenge at KDD Cup 2020. Graph representation learning has been a very hot topic due to ubiquity of graph-structured data, e.g. social network [9], knowledge graph [2], etc. The task of our focus here is node classification under the transductive setting.

Implementation. The AutoGraph benchmark is a typical **code submission** use case. It focuses on AutoML methods which requires more than one dataset to be evaluated together. *Codabench* bundle is by design flexible with **multiple tasks** each of which contains seperate dataset. We also provide a docker hosted on DockerHub, which will be pulled automatically by *Codabench* platform to run each algorithm submission and could also be used for researchers' local development. Every time a new method is uploaded, a new docker container instance will be called to independently run for each dataset. 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[5]. Besides, the datasets are also available to the public for local usage and further benchmarking on Github and Kaggle. To bootstrap the benchmark submissions, we uploaded the solutions of the winners of the challenge. A sample leaderboard can be found in Figure 3. Since the benchmark datasets are released already, users can also run complementary experiments on their local computers and debug mode easily, thus more rapidly making progress. The main incentives to submit to the platform are free hardware and the possibility of showcase results in a common data table.

4.2 Use case 2: DECONbench benchmark

In this section, we introduce DECONbench[4] for benchmarking deconvolution methods inferring the tumor micro-environment composition. We show 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.**

[5]The public AutoGraph benchmark link will be provided later

Task:		Fact Sheet Answers	Dataset a		Dataset b		Dataset c		Dataset d		Dataset f		Dataset g	
#	Participant	Method	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalA
1	xuzhen	3_qqerret	87.35	85.6	75.82	71.6	95.68	91.86	94.51	20.09	92.33	91.06	95.83	94.2
2	xuzhen	1_aister	88.64	87.98	75.61	70.7	96.03	91.92	96.59	51.18	92.68	92.07	95.18	93.3
3	xuzhen	0_baseline	85.8	84.66	71.45	68.01	86.79	72.91	93.62	5	81.34	53.6	94.58	92.7

Figure 3: **AutoGraph benchmark on *Codabench*.** We show here the leaderboard of AutoGraph benchmark. For each dataset, we customize here two metrics as in previous section, accuracy and balanced accuracy. We add extra column for naming the method based on teams to avoid confusion. This leaderboard is set to allow multiple submissions per user to display. On the website, the bottom scroll allows investigation of all the datasets.

Background. Successful treatment of cancer is still a challenge and this is partly due to a wide heterogeneity of cancer composition across patient population. Unfortunately, accounting for such heterogeneity is very difficult and often requires the expertise of anatomical pathologists and radiologists. Therefore, it is pertinent to address this question using computational methods that take advantage of the recent massive generation of high throughput molecular data (called omic data, such as epigenomic or transcriptomic data). DECONbench is a **series of benchmarks** dedicated to the quantification of intra-tumor heterogeneity on cancer omics data, focusing on estimating cell types and proportion in biological samples using epigenomic and transcriptomic datasets (unimodal and/or multimodal). Participants have to identify an estimation of the cell-types proportion matrix underlying the tumor micro-environment composition. 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[6]. Since we are at the same time interested in other modalities of data under similar task, 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[7], immune-cell-types transcriptomic data[8], all-cell-types multimodal transcriptomic and epigenomic data[9].

4.3 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 used for training and inference instead of algorithmic development, which aligns with Data-Centric AI.

Implementation. To solve this question, the COMETH consortium developed the COMETH benchmark [10], a transposed challenge in which datasets should be submitted to be evaluated against existing

[6] https://www.codabench.org/competitions/174
[7] https://www.codabench.org/competitions/147
[8] https://www.codabench.org/competitions/148
[9] https://www.codabench.org/competitions/237
[10] https://www.codabench.org/competitions/218

different reference deconvolution methods (ie "tasks" in the Codabench design) and people can retrieve the corresponding outputs, in a fully reproducible environment. To facilitate the use of this functionality by clinicians who are less familar with data science programming details, COMETH benchmark has been connected to an external client displaying a user-friendly web dashboard. This 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 [11].

4.4 Use case 4: AutoRL benchmark

We lastly introduce another use case: AutoRL 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.

Background. We consider the problem of Dynamic Job-Shop Scheduling. The task is to allocate a set of jobs to a set of machines with stochastic events. Each job has a pre-determined operation sequence to be executed on certain machines. To mimic real life scenarios, we add aleatoric machine down events to the problem. We thus expect an agent policy making decisions on how to schedule better the jobs in minimal time. The reward depends on the makespan.

Implementation. This task can be well formulated into a reinforcement learning framework. As explained in Sec 2, our design of bundle and ingestion/scoring program makes it very natural and flexible for RL problems. We could either follow Figure 2 and use scoring as environment and ingestion as agent, or it is also possible to wrap everything into the ingestion module.

5 Discussion and conclusion

Codabench is a new open sourced platform for data science benchmarks. *Codabench* is compatible with diverse tasks (including supervised learning and reinforcement learning) and 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 scenarios, we introduce 4 benchmark use cases illustrating the flexibility, easiness in use, reproducibility and other features of *Codabench*.

The current limitations of *Codabench* are mainly as follows. First, since it is really new, there are few benchmarks and we do not have yet an active community of organizers and benchmark participants. 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. submit either code or datasets. 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 further.

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 work includes providing more comprehensive usage templates illustrating features such as: (1) splitting an algorithm workflow into sub-modules 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.

[11] `https://cancer-heterogeneity.github.io/cometh.html`

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 source. *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, and ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-00222-01, 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.

References

[1] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor.*, 2007.

[2] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, 2013.

[3] Coleman Cody A., Narayanan Deepak, Kang Daniel, Zhao Tian, Zhang Jian, Nardi Luigi, Bailis Peter, Olukotun Kunle, Re Chris, and Zaharia Matei. Dawnbench: An end-to-end deep learning benchmark and competition. *NIPS ML Systems Workshop*, 2017.

[4] Clémentine Decamps, Alexis Arnaud, Florent Petitprez, Mira Ayadi, Aurélia Baurès, Lucile Armenoult, HADACA consortium, Rémy Nicolle, Richard Tomasini, Aurélien de Reyniès, Jérôme Cros, Yuna Blum, and Magali Richard. Deconbench: a benchmarking platform dedicated to deconvolution methods for tumor heterogeneity quantification. *bioRxiv*, 2020.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[6] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[7] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[8] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the 3rd ACM International Conference on Digital Libraries*, 1998.

[9] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 2017.

[10] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017.

[11] William R. Hersh, Chris Buckley, T. J. Leone, and David H. Hickam. OHSUMED: an interactive retrieval evaluation and new large test collection for research. In *Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[12] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[13] 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. *arXiv preprint arXiv:2005.00687*, 2020.

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

[15] Ken Lang. Newsweeder: Learning to filter netnews. In *International Conference on Machine Learning*, 1995.

[16] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[18] 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*, 12:2825–2830, 2011.

[19] Humphrey. Quill and David. Penney. *John Harrison, Copley medallist and the L.20,000 longitude prize / by H. Quill ; [line drawings by David Penney]*. Antiquarian Horological Society [Ticehurst] ([New House, High St., Ticehurst, Wadhurst, Sussex TN5 7AL]), 1976.

[20] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[21] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, 2018.

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

[23] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR*, 2018.

[24] Quanming Yao, Mengshuo Wang, Hugo Jair Escalante, Isabelle Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *CoRR*, 2018.

Appendix A Codabench usage: getting started

Using *Codabench* as a participant is straightforward. First, create an account and login on *Codabench*. Then choose an existing benchmark to join following the instructions provided by the organizers. To organize a benchmark, a user can either use the *Codabench* editor or upload a benchmark bundle which is a zip file containing code, dataset, and configuration file. Detailed instructions are found on *Codabench* Documentation. For advanced users who wish to deploy a private instance of *Codabench* please refer to *Codabench* deployment instructions in the same wiki. To illustrate better the benchmark bundle, we provide a simplified bundle example in the next section, which contains ingestion program, scoring program, data, text descriptions and a configuration YAML file.

Appendix B Technical aspects of Codabench

In this section, we provide briefly technical implementation details of *Codabench*. *Codabench* is implemented in Python's Django framework, which is one of the most flexible and stable web application framework in Python. The whole system is divided in three main blocks: Front-end, API, and workers. *Codabench* adopts a philosophy of Front-Back separation development. Front-end uses ReactJS, which is open sourced by Facebook in 2013. React largely reduces the interaction with Document Object Model (DOM) by simulating DOM actions. React supports one-way data flow in order to reuse code as much as possible, which makes *Codabench* more extensible. This front-back separation also facilitates the community's maintenance and support. For the API, the Django Rest Framework is used. PostgreSQL database for managing data and MinIO for file storage. In addition, we follow the classic producer-consumer design pattern to allow asynchronous operation, leveraging RabbitMQ as a queue manager, Celery client for message management, Docker for containerization pf *Codabench* itself but also for benchmarks' own consistency (computation workers and customized environment). For more details of the technology implementation, please refer to the *Codabench* Github. Any contribution is welcome.

Appendix C Sample bundle file for Codabench

In this section, we provide a concrete bundle example to show how simple it is to organize benchmarks on *Codabench*. A bundle consists of five parts: (1) a YAML configuration file (2) ingestion program (3) scoring program (4) data (5) text files for additional 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 platform and be displayed on 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 **text files** are just html or markdown files for organizers to provide other information e.g. instructions, references, etc. A final **YAML file** connects all previous parts and provides more configurations for the benchmark. A simplified YAML file is as follows. 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.

```yaml
# Sample bundle based on AutoGraph benchmark
title: 'AutoGraph Benchmark'
description: 'Automated Graph Representation Learning Challenge'
docker_image: nehzux/kddcup2020:v2 # Docker Hub ID
pages:                    # These are "free style" documentation pages
    - title: help         # You can have any title and file name
      file: 'help.html' # You may use HTML or Markdown (.md files)
    - title: overview     # These pages will show up in the benchmark site
      file: 'overview.html'
phases:                   # Benchmarks usually have s single phase
                          # (competitions may have several)
    - index: 0            # Phase order number
      name: 'AutoGraph'
      start: 2021-01-01
      end:   2022-12-31
      tasks:              # Tasks included in this phase
          - 0             # Reference number in task list below,
          - 1             # or absolute reference in Codabench database
      max_submissions: 1000
      max_submissions_per_day: 100
      execution_time_limit_ms: 60000
tasks:                    # Tasks for the above defined phase
    - index: 0
      name: 'Task a'    # For public display on leaderboad
      description: 'Dataset a'   # Private comments
      # Ingestion module:
      ingestion_program: ingestion_program.zip
      input_data: input_data_a.zip
      # Scoring module
      scoring_program: scoring_program.zip
      reference_data: reference_data_a.zip
      # whether the ingestion program is run first, then the
      # scoring program, or the are run in parallel
      ingestion_only_during_scoring: True
    - index: 1
      name: 'Task b'
      description: 'Dataset b'
      # Ingestion module:
      ingestion_program: ingestion_program.zip
      input_data: input_data_b.zip
      # Scoring module
      reference_data: reference_data_b.zip
      scoring_program: scoring_program.zip
      ingestion_only_during_scoring: True
leaderboards:               # Leader board form
    - title: Results        # single leaderboard supported in this version
      key: main             # main key, leave untouched
      columns:
      - title: 'Acc'        # Name of the column displayed
        key: acc            # Data key name used by scoring program
        index: 0            # Order of columns
        sorting: desc       # Sort in descending order
      - title: 'BalAcc'
        key: bacc
        index: 1
        sorting: desc
```

Table 3: Comparisons of AutoGraph competition and AutoGraph benchmark.

	AutoGraph Competition	AutoGraph Benchmark (this paper)
Purpose	Harvesting automated node classification solutions in a short time	Continuous fair evaluation of AutoGraph solutions, over a long time period, in a unified framework to answer multiple qualitative and quantitative questions
Cooperation & information sharing	Limited to submission instructions	Release datasets and meta-data, release challenge winning solutions, forum, Github issues
Submissions	Code with prescribed "fit" and "predict" methods	Code whose execution can produce detailed results in free format
Outcome	Leaderboard with accuracy score for last submission per team	Table with all the submissions made; sorting with 2 scores possible; multiple user-provided analyses, graphs, figures, code sharing

Appendix D More details about AutoGraph benchmark

D.1 AutoGraph Challenge at KDD Cup 2020

The AutoGraph challenge lasted for two months. We received over 2200 submissions and more than 140 teams from both universities (UCLA, Tsinghua University, Peking University, Nanyang Technological University, etc.) and high-tech companies (Bytedance, Twitter, Meituan Dianping, Ant Financial, Criteo, etc.), coming from various countries. The top five teams are: **aister, PASA_NJU, qqerret, common, PostDawn**.

D.2 Benchmark motivation

The motivation of turning AutoGraph challenge into a benchmark are three fold. Firstly, we emphasize the necessity of having a benchmark platform to *fast build domain specific benchmarks* under the exactly same software/hardware configurations. Then we want to demonstrate how easy it is to use *Codabench* to create benchmarks boosting reproducible research. Besides, this benchmark focusing on automated node classification task brings much value to both communities of AutoML and graph representation learning. The AutoGraph competition has its own limitations: strict time constraint, single accuracy metric and last for short period. *Codabench* enables us to fast build a benchmark with more customized metrics, relaxed computational constraints and it never ends. we are now able to allow state of the art methods to be compared in a better way with AutoGraph benchmark. A detailed comparison of AutoGraph challenge and AutoGraph benchmark is illustrated in Table 3.

D.3 Problem formulation

The task of AutoGraph benchmark is node classification under the transductive setting. Formally speaking, consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \cdots, v_N\}$ is the set of nodes, i.e. $|\mathcal{V}| = N$ and $\mathcal{E}$ is the set of edges, which is usually encoded by an adjacency matrix $A \in [0, 1]^{N \times N}$. A_{ij} is positive if there is an edge connecting from node v_i to node v_j. Additionally, a feature matrix $X \in \mathbb{R}^{N \times D}$ gives features of each node. Each node v_i has a class label $y_i \in \mathcal{L} = \{1, \cdots, c\}$, resulting in the label vector $Y \in \mathcal{L}^N$. In the transductive semi-supervised node classification task, part of labels are available during training and the goal is to learn a mapping $\mathcal{F} : \mathcal{V} \to \mathcal{L}$ and predict classes of unlabeled nodes.

In recent years, sophisticated models such as Graph Neural Networks (GNN), e.g. GraphSAGE [10] or GAT [23], have been proposed, leading to the state-of-the-art results in node classification. However, huge computational and expertise resources are needed to achieve a satisfactory performance given a dataset, limiting the application of the existing graph representation models. AutoML [14, 24]/AutoDL[12] is a promising approach to lower the manpower costs of machine learning applications, and has achieved encouraging successes in hyperparameter tuning, model selection, neural

[12] `https://autodl.chalearn.org`

architecture search, and feature engineering. Through this AutoGraph benchmark, we hope to make progress on automated node classification task, which is at the same time challenging and beneficial to practical deployment.

D.4 Benchmark setting

Protocol. The protocol of AutoGraph benchmark is straightforward. Participants should submit a python file containing a Model class with required fit and predict method. We prepare an ingestion program reading dataset and instantiate the class and call fit and predict method until prediction finishes or the running time has reached the limit. Ingestion program outputs model's prediction on test data and save to a shared space. Then another scoring program reads the prediction and ground truth and outputs evaluation scores. When developing locally, we provide script to call model.py file methods directly.

Metric. We use Accuracy (Acc) and Balanced Accuracy (BalAcc) as evaluation metrics, defined as

$$\text{Acc} = \frac{1}{|\Omega|} \sum_{i \in \Omega} \mathbb{1}_{\hat{y}_i = y_i}$$

$$\text{BalAcc} = \frac{1}{|C|} \sum_{i \in C} \text{Recall}_i \ ,$$

where Ω is the set of test nodes indexes, y_i is the ground truth label for node v_i and $\hat{y}_i$ is the predicted label, C is the set of classes and Recall_i is the recall score for class i.

Datasets. Fifteen graph datasets were used during the competition: 5 public datasets were directly downloadable by the participants so they could develop their solutions offline. Five feedback datasets were made available on the platform during the feedback phase to evaluate AutoGraph algorithms on the public leaderboard. Finally, the AutoGraph algorithms were evaluated with 5 final datasets, without human intervention. In the AutoGraph benchmark, we use all datasets from the competition except 2 datasets which can't be published due to IP reason. However, we still provide information about all 15 datasets below. These dataset are quite diverse in domains, shapes, density and other graph properties because We expect AutoGraph solutions to have good generalization ability. We summarize dataset statistics in Table 4. The datasheet, licenses and original sources of these datasets can be found in Appendix.

Table 4: **Statistics of all datasets**. "Avg Deg" is the average number of edges per node. "Directed" and "Weighted" indicate the two properties of a graph. "Skewness" here is calculated by number of nodes in the largest class divided by number of nodes in the smallest class.

Dataset	Phase	Domain	#Node	#Edge	#Feature	#Class	Avg Deg	Directed?	Weighted?	Skewness
a	Public	Citation	2.7K	5.3K	1.4K	7	1.9	F	F	5
b	Public	Citation	3.3K	4.6K	3.7K	6	1.4	F	F	3
c	Public	Social	10K	733K	0.6K	41	73.3	F	F	81
d	Public	News	10K	2,917K	0.3K	20	291.7	T	T	467
e	Public	Finance	7.5K	7.8K	0	3	1.0	F	F	111
f	Feedback	Sales	10K	194K	0.7K	10	19.4	F	F	18
g	Feedback	Citation	10K	41K	8K	5	4.1	F	F	6
h	Feedback	Medicine	10K	2,461K	0.3K	23	246.1	T	T	1,773
i	Feedback	Finance	15K	16K	0	3	1.1	F	F	213
j	Feedback	Medicine	11K	22K	0	9	2.0	F	F	227
k	Private	Sales	8K	119K	0.7K	8	14.9	F	F	6
l	Private	Citation	10K	40K	7K	15	4	F	F	34
m	Private	News	10K	1,425K	0.3K	8	142.5	T	T	360
n	Private	Finance	14K	22K	0	10	1.6	F	F	61
o	Private	Social	12K	19K	0	19	1.6	F	F	62

D.5 Novelty of the AutoGraph benchmark

We compare the AutoGraph benchmark with a widely used benchmark, including node classification tasks: the Open Graph Benchmark (OGB) [13, 12], see Table 5. The biggest novel elements of AutoGraph are: larger number of datasets, multiple simultaneous metrics, automatic code execution in a controlled manner on a platform, comprehensive comparison on ALL benchmark datasets. On

Table 5: Comparison between OGB node classification benchmark [13, 12] with AutoGraph.

	OGB Node Classification Benchmark	AutoGraph Benchmark (this paper)
Scope	Node classification, link prediction, and graph classification	Node classification
Metric	Accuracy (multiclass) or ROC-AUC (binary classif)	Both Accuracy and Balanced Accuracy
Dataset	5 large scale datasets	13 medium scale datasets
Code execution	Not on platform	Code executed on platform, with free resources
Transfer learning	No. One leaderboard per dataset created from harvesting paper results	Yes. Code executed on multi-task benchmark pushing automation

the negative side for OGB, it collects results manually on a per dataset basis with a single metric, and few datasets. On the positive side for OBG, it provides multiple tasks: node classification, link prediction, and graph classification, while AutoGraph focuses only on graph classification.

D.6 Algorithm solutions of AutoGraph benchmark

In this part, we introduce various methods suitable for the AutoGraph benchmark, including AutoGraph challenge baseline and AutoGraph challenge top-3 winners.

Baseline. The baseline is implemented with PyTorch [17] and PyTorch Geometric [6]. In the provided baseline, there is no feature engineering except for using the raw node features. For graph without node features, e.g. dataset i,j, one hot encoding is used to unroll the node lists to a dummy feature table. During model training, a MLP is first used to map node features to the same embedding dimension. Then a two layer vanilla GCN is applied for learning node embeddings. Another MLP with softmax outputs the final classification. Dropout is used. All the hyperparameters are fixed by experience. No time management since the model is very simple and one full training won't cost more than the allowed time budget.

1st placed winner. The 1st winner is from team aister. Their code is open source here[13]. The authors use four GNN models, two spatial ones: GraphSage and GAT, two spectral ones: GCN and TAGC to process node features collectively. For each GNN model, a heavy search is applied offline to determine the important hyperparameters as well as the boundaries. In the online stage, they use a smaller search space to determine the hyperparameters. In order to accelerate the search, they don't fully train each configuration but instead early stop in 16 epochs if the validation loss is not satisfactory. Additional features are used: node degrees, distribution of 1-hop and 2-hop neighbor nodes' features, etc.

2nd place winner. The 2nd winner is from team PASA_NJU. Their code is open source here[14]. They also split the solution in two stages: offline stage and online stage. In the offline stage, the authors train a decision tree based on public data and other self collected datasets to classify graph type into one of three classes. Then they use GraphNAS to search massively optimal GNN architectures including aggregation function, activation, number of heads in attention, hidden units, etc. In the online stage, the authors rapidly classify the dataset and fine tune the offline searched model.

3rd place winner. The 3rd winner is from team qqerret. Their code is open source here[15]. The core model is a variant of spatial based GNN, which aggregates two hops neighbors of a node with additional linear parts for the node itself. Basically, the new embedding of node i is $\hat{h}(i) = \sum_{j \in N_2(i)} a_j h(i) + \alpha(wh(i) + b)$. Additionally, in the GNN output layer, a few features per node are concatenated for final fully connected layer, including number of edges, whether this node connects to a central node who has a lot of edges, label distribution of 1-hop neighbor nodes, and label distribution of 2-hop neighbor nodes.

[13] `https://github.com/aister2020/KDDCUP_2020_AutoGraph_1st_Place`

[14] `https://github.com/Unkrible/AutoGraph2020`

[15] `https://github.com/white-bird/kdd2020_GCN`

D.7 Experiments on benchmark

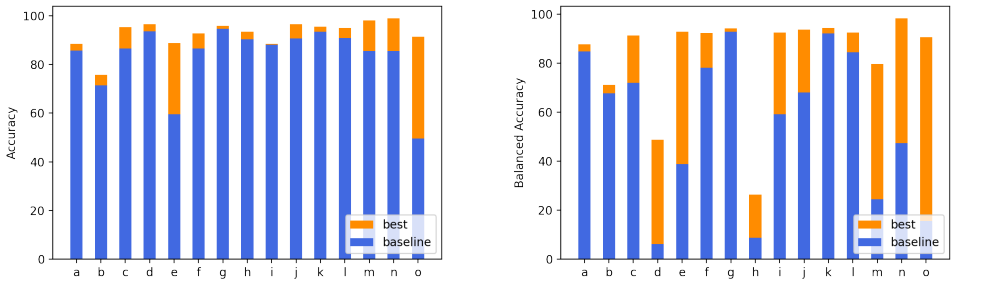
We reproduce all winning methods on all the datasets and include their results in Table 6. The first leaderboard column indicates ranking according the the metric of the challenge (average Acc rank). But, for the benchmark, this is largely irrelevant. Indeed, when we analyse multiple scores on multiple datasets, it is clear that no single method outperforms all the others.

Table 6: **Accuracy and Balanced accuracy of top methods on all datasets (%).** Variances are omitted since all methods fix randomness such that the std is negligeable compared to mean.

Dataset	Phase	Baseline		1st aister		2nd pasanju		3rd qqerret	
		Acc	BalAcc	Acc	BalAcc	Acc	BalAcc	Acc	BalAcc
a	Public	85.7	84.9	**88.5**	**87.8**	88.2	87.2	87.2	85.5
b	Public	71.4	67.8	75.2	69.0	**75.8**	**71.2**	75.6	**71.2**
c	Public	86.5	72.0	94.3	87.5	94.2	90.9	**95.4**	**91.3**
d	Public	93.7	6.1	**96.5**	**48.7**	95.1	28.8	94.6	21.0
e	Public	59.6	38.8	88.7	92.7	88.5	90.7	**88.8**	**92.8**
f	Feedback	86.6	78.2	**92.8**	92.1	92.3	**92.3**	92.4	91.4
g	Feedback	94.7	92.8	95.3	93.5	95.6	93.8	**95.8**	**94.2**
h	Feedback	90.4	8.8	**93.5**	**26.3**	92.2	17.6	92.1	16.6
i	Feedback	88.2	59.2	88.4	87.5	88.4	**92.6**	**88.5**	91.1
j	Feedback	90.7	68.1	95.9	89.0	96.1	**93.7**	**96.6**	93.3
k	Private	93.5	92.2	95.4	94.2	**95.5**	**94.4**	94.8	93.1
l	Private	90.9	84.5	94.9	92.4	94.7	91.8	**95.0**	**92.6**
m	Private	85.5	24.5	**98.1**	**79.7**	95.7	69.0	**98.1**	79.4
n	Private	85.6	47.3	**99.0**	97.3	**99.0**	**98.4**	**99.0**	97.0
o	Private	49.6	15.6	91.0	84.6	91.3	**90.6**	**91.4**	88.5

D.8 Dataset difficulty

In this section, we define and calculate further the concept of dataset difficulty in Figure 4 to retrospect on the benchmark datasets. The intrinsic difficulty is defined as 1 minus accuracy score or 1 minus balanced accuracy score. The modeling difficulty is defined as best performance minus baseline performance. For research interest, it is preferrable to choose datasets of low intrinsic difficulty and high modeling difficulty.



(a) Dataset difficulty based on accuracy (b) Dataset difficulty based on balanced accuracy

Figure 4: **Dataset difficulty measure.**

D.9 License information on AutoGraph datasets

This section provides the license related information on the AutoGraph datasets of Section 4.

Table 7: License for all the datasets

ID	Original dataset	Reference	License
a	Cora	[16]	MIT[a]
b	Citeseer	[8]	CC BY-NC-SA 3.0 [b]
c	Reddit	[10]	MIT[c]
d	20 Newsgroups	[15]	Credit to Ken Lang and Tom Mitchell
e	private	private	Subset of (i)
f	amazon_computer	[21]	MIT[d]
g	coauther_physics	[21]	MIT[d]
h	ohsumed	[11]	CC BY-NC 4.0[e]
i	private[f]	private	AutoGraph challenge dataset, not part of the published benchmark
j	Fresh data	Novel	CC BY-NC 4.0
k	amazon_photo	[21]	MIT[d]
l	coauthor_cs	[21]	MIT[d]
m	R8	Link[g]	Copyright to Reuters Ltd[g]
n	Fresh data	Novel	CC BY-NC 4.0
o	Fresh data	Novel	CC BY-NC 4.0

[a] `https://github.com/kimiyoung/planetoid`

[b] `http://clgiles.ist.psu.edu/pubs.shtml`

[c] `http://snap.stanford.edu/graphsage/`

[d] `https://github.com/shchur/gnn-benchmark`

[e] `https://github.com/huggingface/datasets/blob/master/datasets/ohsumed/ohsumed.py`

[f] We mention datasets (e) and (i) because they are used in AutoGraph challenge. However, due to IP reason, we cannot publicly release them. Thus they are not part of the benchmark.

[g] `http://kdd.ics.uci.edu/databases/reuters21578/README.txt`