PLOS Computational Biology

PERSPECTIVE

Building a continuous benchmarking ecosystem in bioinformatics

Izaskun Mallona[1,2*], **Charlotte Soneson**[2,3], **Ben Carrillo**[1,2], **Almut Lütge**[1,2,4], **Daniel Incicau**[1], **Reto Gerber**[1,2], **Anthony Sonrel**[1,2], **Mark D. Robinson**[1,2*]

1 Department of Molecular Life Sciences, University of Zurich, Zurich, Switzerland, **2** SIB Swiss Institute of Bioinformatics, University of Zurich, Zurich, Switzerland, **3** Friedrich Miescher Institute for Biomedical Research, Basel, Switzerland, **4** Swiss Data Science Centre, Zurich, Switzerland

* izaskun.mallona@gmail.com (IM); mark.robinson@mls.uzh.ch (MDR)

OPEN ACCESS

Citation: Mallona I, Soneson C, Carrillo B, Lütge A, Incicau D, Gerber R, et al. (2025) Building a continuous benchmarking ecosystem in bioinformatics. PLoS Comput Biol 21(11): e1013658. https://doi.org/10.1371/journal.pcbi.1013658

Editor: Francis Ouellette, Montreal, CANADA

Received: November 18, 2024

Accepted: October 24, 2025

Published: November 13, 2025

Funding: MDR acknowledges funding from the Swiss National Science Foundation (grants 200021_212940 and 310030_204869) as well as support from swissuniversities P5 Phase B funding (project 23-36_14). CS is supported by the Novartis Research Foundation. The funders had no role in study design, data

Abstract

Benchmarking, which involves collecting reference datasets and demonstrating method performance, is a requirement for the development of new computational tools, but also becomes a domain of its own to achieve neutral comparisons of methods. Although a lot has been written about how to design and conduct benchmark studies, this Perspective sheds light on a wish list for a computational platform to orchestrate benchmark studies. We discuss various ideas for organizing reproducible software environments, formally defining benchmarks, orchestrating standardized workflows, and how they interface with computing infrastructure.

Introduction

Benchmarking (defined below) is a critical step in method development. Such studies are typically published as either 'methods-development papers' (MDPs) where new methods are compared against existing ones; or 'benchmark-only papers' (BOPs), where a set of existing methods are compared in a more neutral way, as defined in [1]. A benchmarking study can be run by a single person (solo benchmarking) or a small group, or it can be organized as a 'challenge' or within a hackathon. In this work, we present a perspective on building a robust benchmarking ecosystem in bioinformatics, coming primarily from the methodologist's viewpoint (researchers developing computational tools). While our focus is bioinformatics, we aim to abstract principles and connect to similar challenges in adjacent fields.

In our discussion, we outline concepts related to workflow automation, benchmark formalization and benchmarking-specific tasks beyond the former, including philosophical issues like gatekeeping and trust; and provide some technical recommendations, such as standardization.

Why a benchmarking system?

We define a benchmark as a conceptual framework to evaluate the performance of computational methods for a given task, though it can extend to assessing other

collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

aspects, like data-generating technologies. A benchmark thus requires a well-defined task and typically a definition of correctness, or ground-truth, to be precisely defined in advance (Fig 1). A benchmark then consists of benchmark 'components', such as simulations to generate datasets, preprocessing steps, methods, and metrics.

Can benchmarking be systematized? Benchmarking aims to calculate, collect and report performance metrics of methods aiming to solve a task. Such a process normally requires collecting appropriate data and running methods in an automated manner. Some of these tasks can be automated by workflow systems. Others, such as sharing results in an interactive manner and, particularly, result interpretation and making results reusable by others, go beyond what workflows offer. Similarly, collaborative benchmarking and challenges add extra layers of complexity, including collaboration tracking, hiding ground truths, or the ability to release multiple versions of the same benchmark as it develops (Fig 1). Moreover, scientific benchmarking borrows procedures from the scientific method and from scientific publishing, hence having expectations of fairness, reproducibility, transparency, and trust. *Benchmarking systems* aim to orchestrate the workflow management, benchmarking and community engagement aspects of the process to generate benchmark 'artifacts' (code snapshots, file outputs to be shared, performance outputs) in a systematic way and following good practices and standards. Namely, the various elements of code, especially the details on how datasets are preprocessed and how methods are run, should be explicitly available for scrutiny. Such systems should facilitate components to be public, open and allow contributions (e.g., corrections or new components), such that the benchmarks of the future can run more continuously. At the outset of a benchmark, a benchmark 'definition' could be envisioned, which could give a formal specification of the entire set of components (and the pattern of artifacts to be generated). The benchmark definition could even be expressed as a single configuration file that specifies the scope and topology of components to include, details of the repositories (code implementations with versions), the instructions to create software environments (accessible across compute architectures), the parameters used, and what components to snapshot for a release. A benchmark definition in terms of layers and the corresponding challenges and opportunities within each layer is illustrated in Fig 2.

Benchmark stakeholders

Benchmarks are useful for a broad set of stakeholders. For a *data analyst*, with an aim to perform a specific type of analysis on a specific dataset, benchmarks are helpful tools to find a suitable method for the task. Hence, it is important that benchmarks include datasets similar to the one of interest to the analyst, as method performance often depends on characteristics of the data. Thus, a thorough characterization of the datasets considered in a benchmark is desirable. Given that benchmarks typically evaluate methods using a range of metrics and datasets, not all of which are relevant to all analysts, flexibility in defining the ranking approach and aggregation of metrics is preferable. Such flexibility is typically not part of published benchmarks today [2].

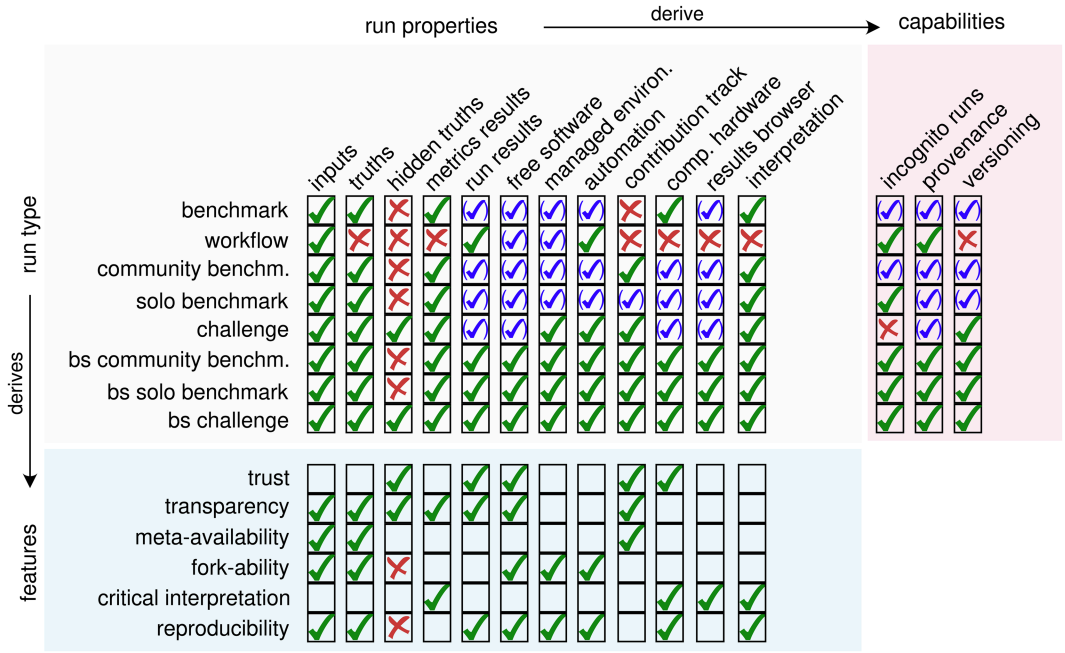

Fig 1. Solo and community benchmarks, workflows, and benchmarking-system-enabled (*bs*) benchmarks all differ in run properties and their consequences in terms of capabilities and features, including whether: inputs, truths, hidden truths, metrics or run results are available for readers; code is free and open source software, software environments are managed, and execution automated; there is a clear track of (human) contributors; results are run in comparable hardware; an entrypoint to results or result interpretation are provided. These run-intrinsic properties have consequences in terms of features (i.e., transparency, forkability, availability for metaanalysis) and technical capabilities (i.e. software versioning or ability to run locally without leaving traces - in *incognito* mode). Green, blue and red ticks depict available, perhaps available and unavailable features, respectively.

https://doi.org/10.1371/journal.pcbi.1013658.g001

However, a well-structured benchmarking system could accommodate flexible filtering and aggregation of metrics and datasets, and additionally provide access to the code and software stack that the analyst would need to apply the selected method to their own dataset.

For *method developers*, benchmarking allows their method to be compared to the state of the art, ideally using neutral datasets and metrics to avoid intrinsic bias [3,4]; a recent call for reporting checklists was also made [5]. A diverse, accessible set of benchmarking datasets and metrics can address this bias and lower the entry barrier for method developers. Moreover, it is important that the new method is compared to the current state of competing methods, to best mimic the choices available to a potential method user. Alternatively, new methods could be compared to previous versions or even different implementations of the 'same' method [6], to illustrate stability of code changes or improvements made over time. In both cases, the typical approach today would be for the method developer to rerun all the methods and versions to which they want to compare their current method, on all the datasets they wish to use. This can be challenging and time consuming, since not all methods run on the same hardware setup or in the same software environment (e.g., R and Python versions, library dependencies). It also leads to a lot of redundancy across method comparisons (multiple stakeholders implementing similar workflows), which could be reduced by the presence of a benchmarking system where such results would be accessible and extendable. Finally, the results of the benchmark of a new method should be included in the corresponding publication, and thus an easy way of including the new method into the set of results served by the benchmarking system, and generating a snapshot in time for reproducibility, would be of great help to the developer.

Benchmarks can also be of high value to *scientific journals* and *funding agencies*. Well-executed benchmark studies can be highly utilized and influential, and may point to gaps in the current body of methods, guiding future methodological developments; at the same time, in fast-moving fields like bioinformatics, benchmarks have the tendency to rapidly

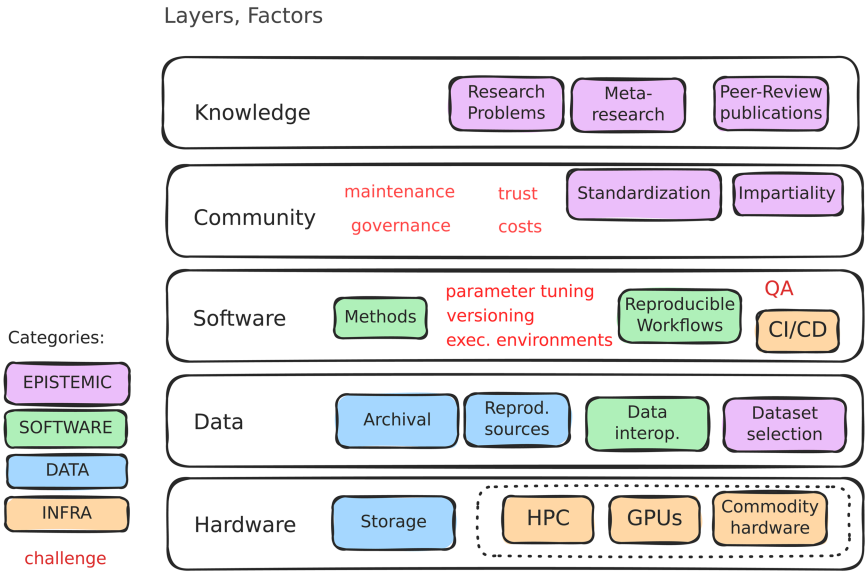

Fig 2. **Benchmarking conceptual layers and their challenges.** The process of benchmarking is multilayered, with layer-specific factors belonging to different categories (colored boxes) and challenges (in red). *Hardware*: infrastructure and its costs. *Data*: dataset archival, openness, interoperability, and selection. *Software*: method implementations, reproducibility, workflow execution, continuous integration and delivery (CI/CD), versioning, quality assurance (QA). *Community*: Standardization and impartiality (governance, transparency, building trust, long term maintainability). *Knowledge*: research and meta-research, academic publications.

https://doi.org/10.1371/journal.pcbi.1013658.g002

become stale. Journals and funding bodies also have a vested interest in ensuring that published or funded method developments are performed to a high standard, that unnecessary redundancy is reduced, and that results are FAIR (findable, accessible, interoperable and reusable) for maximal benefit to the community [7]. Recently published benchmarks and challenges have been conceived as or ported to benchmarking systems to ensure these properties [8–10]. Thus, quality assurance, neutrality and transparency of benchmarking systems become highly relevant.

Collaborative benchmarking efforts enunciate field-specific challenges and provide a venue for *communities* to address them. For instance, DREAM [11], GIAB [12] and CASP [13] challenges and benchmarks have demonstrated how collaborative benchmarks can help in moving fields forward.

Finally, benchmarks could form a research topic on their own, either for newcomers to the field to get acquainted with the conceptual options available, or for experts to consolidate research efforts of a subfield. For this, it is important that there are suitable venues for publishing benchmarks. Regardless, a *benchmarker* (i.e., researcher leading a benchmarking study) would also benefit from using a benchmarking system and curated collections of benchmark artifacts. Such benchmarkers may be good candidates for curating and maintaining such collections by designing and leading benchmark efforts as well as guiding other contributors to adhere to a high standard.

Benchmarks can be formally defined

Benchmarks are collections of data and source code, ideally executed as workflows within a computing environment. Indeed, recent studies highlight that code is frequently open, reusable and versioned, as are input datasets [1,2]. However, *extensibility* of benchmarking studies is generally very low, as is the proportion of benchmarks using a formal workflow system.

Over 350 workflow languages, platforms or systems [14,15] and one standard (Common Workflow Language, CWL) exist [16]. Some workflow languages can export workflow definitions to CWL, hence helping computational FAIR guidelines, i.e., bundling and processing data together with their metadata, tracking new metadata generation, recording provenance, and being accessible [17]. For benchmarking, we believe both the workflow layout and provenance can be formally defined (Fig 3).

Benchmarking as a process deals with tasks other than workflow definition and execution, including: collecting and keeping track of contributions from users; provisioning appropriate hardware; handling a reproducible and efficient software stack (the advantages of decoupling environment handling from workflow execution are further discussed in the section 'Reproducible software environments'); managing storage and access-control lists; rendering results dashboards and summarizing results; versioning code, workflow runs, and files; and providing extensive documentation from defining scope or contribution guidelines, to transparent logging and crediting contributions and disseminating results (Fig 1).

Currently, most benchmarking systems aim to enhance the workflow formalization and execution with strategies to fulfill benchmarking-specific needs. To name a few of these systems, `ncbench` bundles workflow specification and software management with `Snakemake` to benchmark result visualization with `Datavzrd` [18]. `OpenEBench` runs workflows in any workflow language, frequently `nextflow`, and makes use of `openVRE` as GUI and means of visualization [19]. `OpenProblems` uses `Viash` [20] and `nextflow` to handle software environments and workflows, respectively, and reports updated leaderboards with custom code [21]. To our knowledge, no benchmarking platform makes use of a fully expressive benchmark definition formalization language covering workflow generation and benchmarking-specific tasks. Orthogonally, some components of the benchmarking process have been formalized: `Viash` [20] systematizes workflow components; `PROV-O` represents provenance data [22]; and `Research Objects` (`RO` [23], e.g., `RO-Crate` [24])

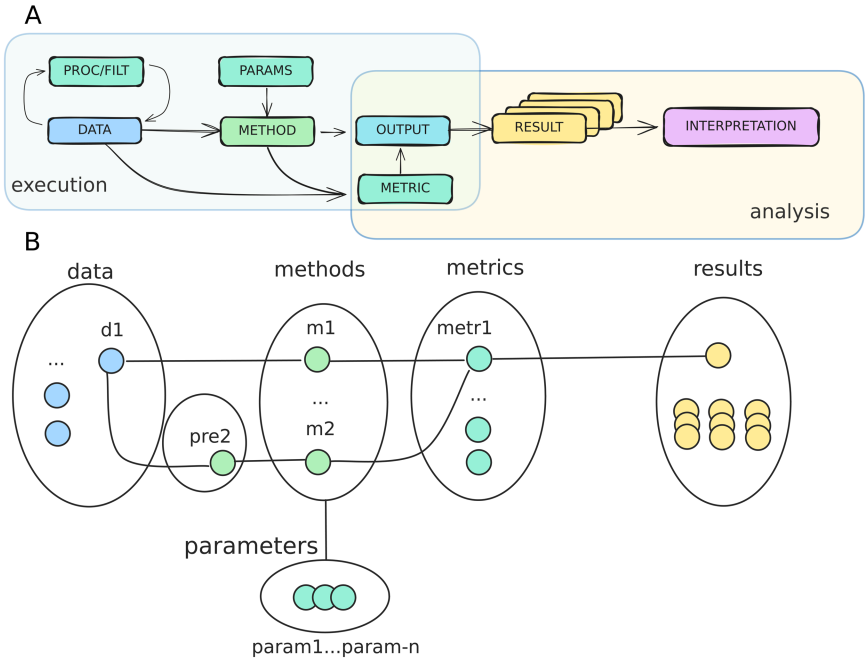

Fig 3. Benchmark formalization. A) Running a benchmark requires running a workflow (execution phase) to generate results to be critically evaluated (analysis phase). **B**) The execution phase consists on a mapping of methods to specific input files to generate output files, with optional steps to harmonize data and parametrize runs.

https://doi.org/10.1371/journal.pcbi.1013658.g003

specify a system to store data (files) and their metadata. Reusing pre-existing formalizations and extending them to the benchmarking field would facilitate conceptualization, community gathering and result sharing.

Scope and interpretation of benchmarking results

Benchmarks are often focused on finding the 'top' performing method(s), or on generating rankings. While this is valid in a few fields with clearly defined goals, such as image recognition [25] or protein structure prediction [26], bioinformatics tasks are typically evaluated by multiple metrics, some of which are not well understood [27,28]. Additionally, ground truths are not always well-defined (e.g., derived with uncertainty from other experimental data) and multiple tasks (e.g., represented by different reference datasets) may be evaluated for a single method and several ways to assess performance are possible. Even when accompanied by dataset- and/or metric-specific performance measures, scaling and aggregation of results make it difficult to unpack fine-grained nuances of performance from a final summary table reported as a FunkyHeatmap [29] as in most benchmarks (e.g., [30]). Strobl *et al.* push back against the 'one method fits all datasets' philosophy and advocate for highlighting dataset or parameter properties that are predictive of method performance [31–33]. This aligns with what a data analyst wants: to identify the best performers in settings most similar to their problem at hand. This requires not only a thorough benchmark design, e.g., tracking of meaningful dataset metadata [33], but also can be supported by a benchmarking system that facilitates flexible navigation of results.

Besides interactive dashboards for more nuanced result exploration [34], multi-criteria decision analysis (MCDA) [35] has been proposed to guide users through benchmark results and multidimensional scaling (MDS) has been used to differentiate the effect of single datasets or metrics and recognize patterns beyond aggregated performance [36]. An extensive analysis and presentation of the results helps not only users in the long term but also the whole scientific field to identify limitations across various use cases. The first advantage of such a strategy is to force the user to an attentive reading of the performance results so that the correct use case and relevant methods can be identified. The second, more important, advantage of presenting benchmark results extensively is to identify the nuances, e.g., the scenarios, types of datasets, or aspects of performance that are systematically associated with poor results. Being public and open to contributions from specialists of different fields, a benchmarking system could gather more reference datasets and methods, and allow the current state of the field to be better grasped.

Parameter choices influence results interpretation, as they can be included in a benchmark to test performance across a wider range of settings, or to find the optimal parameter settings. Dataset-specific parameters, e.g., the true number of clusters when evaluating clustering algorithms, are not method-specific; they can be seen as (dataset-specific) metadata. To differentiate between the effect of the parameter and the methods performances themselves, data-specific parameters should be shared across methods during evaluation. Similarly, optimizing method-specific parameters for some but not other methods can skew results and should be avoided. Aside of optimal and method-specific parameter searches, comparing a method under different settings against other methods can be viewed as comparing fully distinct methods, which should be clearly communicated in the results. For continuous and collaborative benchmarking, parameter optimization and usage should be governed and coordinated between all contributors, ideally ensuring comparability and interpretability of the results.

Aside of task-tailored scores (i.e., intrinsic or extrinsic cluster validity measures for clustering tasks), collecting metrics to report computing performance (i.e., wallclock, on-disk read-write operations), requirements (e.g., GPUs, minimum memory) and their scalability depending on the input size (e.g., whether runtimes increase linearly, quadratically, etc.) is key during benchmarking. To measure these performance metrics accurately, benchmarking systems ideally would provide means to: i) characterize the hardware running the benchmark, so method runtimes are comparable; ii) track performance (CPU or GPU usage, memory) when running each benchmark component; iii) introduce minimal overhead during profiling; iv) collect variability in performance by running each method several times; and, v) go beyond reporting average

resource usage and provide them over execution time (e.g., so embarrassingly-parallel algorithms can be differentiated from largely single-cored runs with few bursts of parallel execution).

Collaboration, gatekeeping and benchmarking infrastructure go hand in hand

In designing a benchmarking system, it may be helpful to specify whether the benchmark is intended to be run 'centralized' or 'decentralized'. These terms can take different meanings depending on the context, but here the benchmark infrastructure location is intended. In most systems, code will be tracked in a `git` repository (e.g., on GitHub, GitLab). The workflow execution and the storage of derived files are typically located either in local deployments (e.g., `OEB-VRE` for `OpenEBench`), commercial clouds (e.g., AWS for `OpenProblems`), commercial continuous integration and continuous delivery (e.g., GitHub actions for `ncbench`) or on arbitrary infrastructure (i.e., laptop, server or cloud; e.g., omnibenchmark [37]).

Traditionally, benchmarks (whether BOP or MDP) are conducted in a solo and decentralized fashion, being started by a single researcher or small team (e.g., for MDP, the new method developer) using their local infrastructure. This brings some disadvantages. For example, decentralization can be hugely inefficient; researchers studying the same task would collect reference datasets and code snippets to run methods, and implement a bespoke strategy (e.g., shell scripts) to orchestrate the methods run on datasets without being aware that their colleagues are simultaneously undertaking similar efforts, leading to redundant and fragmented work. Working together would mean sharing not only datasets and code snippets, but also execution environments (software but perhaps also hardware), alignment on parameters to evaluate and eventually sharing, consolidating, and interpreting results. However, because there are not many standards of how benchmarks are implemented [2], another disadvantage of decentralized benchmarks is that, while these components may eventually be openly shared as a product of a research study, the contents of a benchmark are non-interoperable, even across benchmarks of the same task. Furthermore, decentralized benchmarks have been criticized on the basis of 'inflating model performance' due to decisions made in preprocessing or parameter tuning [21]; such issues could be mitigated by gathering the 'wisdom of crowds' that is implicit in running centralized benchmarks.

We believe most benchmarks begin decentralized, allowing for concept definition and preliminary results as conducted by solo researchers or small teams before 'going public', i.e., welcoming contributions from other researchers while also agreeing on common practices, standards and a jointly accessible infrastructure for compute and data and code sharing; we call this the centralization phase. Users need a system that supports building benchmarks both ways, transitioning to centralization as the design matures. However, becoming centralized is not cost-free: it implies gatekeeping (contributors authenticate to the system and get access to computational resources) and requires additional measures that build trust (e.g., transparency).

Compute-intensive benchmarks are often run using cloud or high performance computing, further extending the challenge to ensure hardware homogeneity (for comparable performances) and adding complexity to the benchmarking system capabilities to automate node provisioning (or node selection) and to group executions such that costs are minimized. Some workflow managers, such as `Snakemake`, provide plugins to execute on the cloud or on-premises queuing systems, effectively decoupling workflow specification and execution.

Building trust and bringing together communities

An active community is vital to the success of a benchmarking system. Benchmarkers (which may include multiple scientists) are responsible for planning and coordinating a benchmark, defining the task, possibly splitting it into subtasks or processing stages, and defining the data formats across the stages; the benchmarker also brings an authority role of how to review and approve contributions. Contributors (which may include method developers) curate and add content to the benchmark, which could be new datasets, methods or metrics, adhering to the guidelines set up by the benchmarker. Finally, the viewers (which may include benchmarkers and contributors) of benchmark results are users who retrieve one

or more of the artifacts (code snippets, dataset files, intermediate results, or metric scores). These could span a range of use cases, including the data analyst choosing which method to use for a specific application, an instructor retrieving a curated dataset for teaching purposes, or a methods researcher prototyping their new method.

For assembling and sustaining communities, strategies include organizing seminars or workshops (on the educational side) or hackathons and challenges (aimed at practical onboarding or contributions). Career-related or financial incentives, such as the prospect of a scientific publication, or compensation for the allocated time, are also common motivators. For a method developer, contributing to a public benchmark provides a way to test their method without the need to set up all the infrastructure themselves, and also provides an opportunity to advertise their work. Ultimately, the likelihood that someone will contribute to a benchmark will be associated with whether the results are of interest to them, as well as how easy it is to contribute. Contributions can be made easier by providing good organization, suggested 'good first issues' to tackle, contributor guidelines and a code of conduct.

Authorship, e.g., acknowledging contributions, can be captured, reported and credited using third-party platforms such as Apicuron [38] or by bundling standardized contribution metadata such as the Citation File Format (`CFF`) [39]. In this way, individual contributions to a benchmark can be acknowledged, browsed, and ideally propagated to academic deliverables (e.g., datasets, publications).

Publicly soliciting contributions poses challenges, requiring a transparent setup to build trust. Benchmarkers must trace and test contributions to detect misuse and prevent non-functional or suboptimal implementations. This can be addressed by a 'quarantine zone', where new content is automatically tested before being added to the system. Similarly, a contributor should be able to trace their contribution, to ensure that no unintended modifications are made after submission. Ultimately, many risks can be mitigated by transparency [40], and feedback loops with (all) developers.

A key challenge is selecting datasets and metrics for a benchmark. Diversity of these ensures generalizability and prevents overfitting, but low-quality or redundant datasets can lead to misleading results. Possible mitigations include sensitivity analyses, evaluating the impact of single datasets or metrics on overall performance metrics, or reducing redundancy by down-weighting similar metrics to avoid a specific aspect dominating the evaluation. From a wider perspective, these questions are all related to the governance of a benchmark, a topic that has not received much attention. For governance, benchmarkers could choose to have the ultimate authority to decide on what gets included in a benchmark, simplifying the decision making but creating a risk of gatekeeping; or, decisions could be made collectively by a council of contributors, which may mitigate the risk of gatekeeping at least partially, but creates a more complex decision structure and introduces the need for a conflict resolution strategy. The governance model of a benchmark could also cover aspects related to computing infrastructure and storage, as well as guidelines to manage authorship of any resulting publications or resources. Having the governance model spelled out early in a project is likely to reduce friction later.

Neutrality is a highly desirable property in a benchmark: it means that efforts have been taken to ensure impartiality or unbiasedness (or at least transparency) of the various choices made (e.g., what datasets to include, how exactly to run methods, how to evaluate methods) [4,31]. One could claim that all MDP benchmarks are non-neutral, since the goal is generally to highlight the virtues of a new method. Similarly, ground truth generation and simulations in MDP benchmarks can unconsciously mirror the method capabilities, mechanisms, or strengths, thus biasing evaluation. Furthermore, there are many field experts who may wish to conduct a benchmark (or participate in a collaborative benchmark) on analyses that involve their own tools, where neutrality is then harder to establish. Here, recent work to mitigate this conflict involves pre-registration and posting the parameters of the study in advance at a registry [41–43]. In contrast, BOP benchmarks give, in principle, a higher degree of neutrality, but the challenge to assess neutrality directly remains. Other benchmarking models are possible, from challenges to hackathons, with the hope that a consensus of the masses can further help to establish neutrality. One potential strategy, in the context of the systematic design of benchmarks, is establishing a system where a benchmark definition is parsed into a pre-registration template. Templates exist for simulation studies that could be adapted more generally for benchmarking projects [44].

Transparency and clear governance guidelines provide an effective means to detect and react to potential misuse, e.g., by explicit rules to validate results. Other ethical considerations include facilitating safe interactions and fair resource allocation and efficiency. Clear code of conduct statements and proactively flagging harassment (e.g., by sex, ethnicity or academic seniority, among others) provide explicit means to ease collaboration. In terms of efficiency, comparable allocation of resources (e.g., computing power, support) to all contributors, while also being aware of CO_2 emissions, could also be spelled out in the governance model.

Mature communities are key to sustaining and reflecting the existing challenges in a given field. For instance, CASP, a continuous benchmark by nature, has a established community that defines the scope of critical assessment, input data, target definitions, and composite and normalized metrics to equally weight multiple performance scores [45]. At the continuous benchmark level, CASP keeps track of past performances and introduces metrics to evaluate possible artifacts, i.e., whether progress during a CASP round could arise from different target difficulty [46]. To facilitate community contribution, we believe that benchmarking systems can lower the barrier to contributing, hence making the contribution efforts less likely to be biased.

Finally, there is a tension between preregistration and the concept of 'continuous' benchmarking, where new datasets, methods and new metrics are added over time. The logic of continuous benchmarking is that as the computational task becomes more understood, it becomes increasingly clear how to best evaluate the performance of methods.

Long-term software reproducibility and data preservation

Software plays an important role in the current scientific landscape [47], since it is widely used throughout the scientific process, including data collection, simulation, analysis and reporting. Even if open sourced, however, it is sometimes easier to develop new code than to reuse existing software [48]. This leads to the phenomenon of 'academic abandonware', where projects are forgotten in code repositories and not maintained after delivering a research output.

Operational costs of benchmarks are given by the expected compute and storage requirements. In the context of benchmarking, methods under evaluation typically have variable computational demands. Benchmark designers could impose resource limits, thereby evaluating the methods under constraints. Storage costs, on the other hand, are more predictable and can be directly influenced by design decisions of the benchmark (e.g., low retention for re-generatable artifacts, 'cold storage' for code archives and software, external data stores such as Zenodo for method performance artifacts, CernVM-FS for reusable software).

Design properties

A good benchmarking system needs to package its components in a way that ensures that their execution and artifacts can be consistently replicated (within practical limits).

The UNIX design philosophy [49] is worth embracing to produce software for benchmarking systems that are lightweight, robust, and easily maintainable; ease of use and extensibility are also desirable. Furthermore, to reduce the entry barrier for contributors, the benchmarking system should be capable of running locally with minimal setup. This approach ensures that the system remains relevant and functional even in the absence of centralization. Furthermore, intuitive APIs and sensible defaults [50] are factors that contribute to highly usable systems. Structured approaches to documentation, including frameworks such as diataxis [51], can facilitate adoption and new contributions to a benchmark.

Reproducible software environments

Computational reproducibility comes from controlling the triad of data, code, and environment [52]. During benchmarking, a workflow runs code transforming inputs into outputs; execution is affected by the base system (operating system, compiler toolchains, libraries) and its configuration. The benchmark definition should control the execution environment.

Leaving data aside, it can be useful to divide the codebase used into three distinct categories:

1. The benchmarking system has an impact on operational costs, such as storage needs, execution platform, etc. The system mandates the choice of a particular stack and workflow manager. At this level, any requirements for input/output formats and shapes are defined.
2. Benchmark components, like dataset-generating simulations, methods, or metrics, are typically small scripts that process data and use external libraries. Even if method development happens elsewhere, good practices should be embraced (output validation, abnormal termination checks, running on subsets of input data). Metadata validation should be enforced for each module, including authorship, versioning, and licenses.
3. Software dependencies cause the biggest impact on replicability and maintainability; archiving all combinations of a large dependency tree, across several OS images, exponentially increases retention and maintenance costs. Making the long-term replicability of a benchmark tractable implies adopting a sane software management system.

The responsibility of handling software environments poses an interesting challenge to benchmarking systems. On one hand, some workflow managers provide means to install and activate software and re-run workflows when software specification changes, even if the workflow topology remains the same. For these, software is the workflow manager's responsibility. On the other hand, some benchmarks might be designed to compare the same workflow topology run with different software backends (i.e., with `conda` vs compiled) or with different versions of a dependency (e.g., `samtools`/`htslib` releases). Benchmarking systems should provide enough flexibility to make all variants possible (i.e., decouple the benchmark topology from the software backend).

Finally, scientific software management handling is known for its nuances, including abundant dependencies with known incompatibilities and complex installation recipes [53]. Scientific software management is handled via traditional (Linux) package managers (RPMs, DEBs), automated and reproducible installs (such as `EasyBuild` [54] or `Spack` [55]), central software stacks and environment modules (i.e., `Lmod`), `conda`, containers (e.g., `apptainer` or `Singularity`; [56]), or read-only network filesystems providing ready-to-use installations (`CernVM-FS`) [57]. Further strategies are under active development: EESSI [53] and the Digital Research Alliance of Canada [58] are effectively providing multi-purpose, reusable and efficient software installations via `CernVM-FS`; `conda` is speeding up their deployment procedures (via `py-rattler`/`pixi`) and allowing compilation to Web assembly (to execute code within a web browser); and `apptainer` (former `Singularity`) provides rootless containerization suitable for running in scientific clusters.

Hardware

Software, both from benchmarking systems and from the benchmarking workflow itself (the benchmarked methods), can be run on different hardware, from commodity computers to dedicated hardware, often large servers or academic and commercial clouds. Performance profiling (e.g., how fast a benchmarked method runs) requires access to fair and comparable resources.

Comparable resources can be provisioned on demand in cloud computing, or can be ensured by deploying benchmarking systems in controlled and dedicated hardware. Ideally, performance results (e.g., wallclock time) could be bundled to metadata describing the used hardware (e.g., CPU characteristics).

It is worth noting that CPU microarchitecture influences performance, particularly when using optimized compilers to improve performance of benchmarked methods at the expense of making binaries microarchitecture-specific. This feature is not solved by containerization. Benchmarking systems can automate software installations with equivalent degrees of optimization across benchmarked methods, and/or provide multi-architecture builds.

Finally, benchmarking methods could benefit from high performance computing-style user quotas and timeouts to cap the maximum memory and/or CPU cycles each method can use.

Software licenses and attribution

Even a simple, single-person benchmark comparing methods relies on multiple contributions, from dataset creators and method developers to system software and workflow authors. Hence, data, code and operating system dependencies are all accessible during the benchmarking process. For reproducibility, transparency and to build trust [59], this accessibility and freedom should be extended to others to run (re-run), copy (including fork), distribute (e.g., snapshot), study (e.g., read how someone is implementing a method), change (bug fix, contribute) and improve (extend) benchmarks. Such freedoms are granted (or restricted) by software and content licenses, as are copyright and authorship and attribution [60]. Without explicit licenses, even publicly available code is copyrighted by default, and reuse is restricted (to reproduce, distribute, or create derivative works) [60]. A concise summary of license rights and restrictions, and compatibility across commonly-used open-source licenses can be found in Table 1. Benchmark contributions should be licensed and associated to scholarly names including email and ORCID (as the canonical identifier), hence allowing seamless reuse, transparency and attribution. As a consequence, properly-licensed benchmark components (e.g., a procedure to compute a clustering metric; or a clustering method) could be reused across benchmarks; we envision assisted benchmark design tools to leverage and reuse benchmark components, perhaps guided by AI.

Similarly, even if all benchmark code were free and open source software (FOSS), running it could be hindered by (non-free) proprietary interpreters (e.g., MATLAB). Benchmarking systems should be explicit in allowing or disallowing proprietary software, and be aware and track EULAs (end user license agreements). This conundrum extends to the operating system layer and to system dependencies (e.g., compilers). In terms of benchmark re-runnability, fully FOSS system dependencies arguably offer the highest reusability, and can be handled in a reproducible manner across operating systems and system architectures (e.g., amd64, arm64) [53]. In contrast, proprietary software often restricts redistribution and re-execution due to licensing constraints, making it inherently non-rerunnable in many benchmarking contexts. Hence, benchmarkers should evaluate the trade-off between including all possible software, including proprietary, and its associated limitations for reproducibility and long-term accessibility.

Finally, data files (inputs and outputs) are frequently stored in centralized repositories [61–63] under content licenses, which might enforce attribution or copyleft. A benchmarking system should automate license metadata handling during data import and export, crediting the original authors if applicable; it also represents an opportunity to enforce that academics use appropriate licenses for data and code.

Table 1. **Overview of commonly used open-source licenses in benchmarking studies.** Code or data with restrictions to redistribute or modify are not suitable for collaborative benchmarking. *License*: license name. *Redistribution*: right to redistribute, e.g., reuse in another benchmark/somewhere else. *Modification*: right to modify the code to produce derivative works, e.g., in another benchmark/somewhere else. *Sublicensing*: for derivative works, obligation to keep the same license or freedom to be licensed differently (even copyright). Relevant if the code contributor does not want derivative works to be 'closed source' or copyrighted. *Commercial use*: for derivative works, whether commercial usage is restricted or not. Relevant to code contributors not willing commercial reuse of their code (by others). *Attribution*: for derivative works, obligation to attribute original author(s). Relevant to code contributors willing their names/authorship to be credited in derivative works. *Linking*: dynamic or static linking / use as libraries. Particularly important for metrics, as these code can be frequently incorporated (=executed) across benchmarks.

License	Redistribution	Modification	Sublicensing	Commercial use	Attribution	Linking
no license	not allowed	not allowed	no sublicensing	not allowed	required	not allowed
GPLv3	allowed	with restrictions	GPLv3-compatible	allowed	required	GPLv3-compatible
Apache	allowed	allowed	minimal restrictions	allowed	required	allowed
MIT	allowed	allowed	full freedom	allowed	required	allowed
Public Domain	allowed	allowed	full freedom	allowed	not required	allowed
CC-BY-NC-ND	allowed	not allowed	(no derivatives)	not allowed	required	(not a code license)
CC-BY-SA-NC	allowed	allowed	not more restrictive	not allowed	required	(not a code license)
CC-BY-SA	allowed	allowed	not more restrictive	allowed	required	(not a code license)
CC-BY	allowed	allowed	full freedom	allowed	required	(not a code license)

https://doi.org/10.1371/journal.pcbi.1013658.t001

There are various hidden design tradeoffs

So far, no single benchmarking system can accommodate all use cases in bioinformatics. However, documentation of design choices for a given system can help to align expectations. Some important design trade-offs are:

- Flexibility vs. constraints: An unconstrained benchmark grants method contributors greater flexibility but requires more effort from the benchmarker (maintenance, version support). Introducing constraints (such as software dependencies or common data formats) adds initial setup work but can boost maintainability through built-in validation and quality checks.
- Achieving software reproducibility needs discipline and commitment [64]. A benchmarker can state the desired degree of replicability and constraints.
- Security concerns: decentralized runs lower the participation barrier, but also increase the attack surface. Sandboxed environments (restricting network access, filesystem access etc.) can be provided as mitigation. Additionally, static and dynamic code analysis in the continuous integration process can aid in approving potentially risky submissions.
- Versioning granularity: a single benchmark could address performance of a given method across its versions when run on the same datasets; independently, this benchmark could be updated so several benchmark versions are released. Benchmarking systems should provide versioning at both component and benchmark levels.

Open data formats and standards

Benchmarking involves applying methods to data. Data types can be diverse and even a single dataset can be stored in multiple formats. A broad distinction can be made according to general vs. specific formats and open vs. proprietary formats. File formats specific to a programming language should be avoided since those formats may change over time, security risks can arise [65,66], and they are not always interoperable. Likewise, proprietary formats should be avoided, since they may change and could require specific (proprietary) software to read. General language-agnostic formats often do not change over time and thus align to FAIR principles [7]. Generally, using established standards is recommended (e.g., SAM, BED, FASTQ, etc. for genomics), since related methods are developed around these formats.

If raw input data consists of multiple file formats, an option could be to convert the data into a common format with the caveat that this 'intermediate' raw data could be either stored (increased storage usage, easier benchmark definition) or created 'on the fly' (increased compute). Furthermore, how raw input data are retrieved (e.g., automatic retrieval once and store locally, automatic retrieval every time, or manual retrieval and store locally) needs to be decided. These intermediate results could also be made FAIR.

Additionally, validation is recommended. Validations include content integrity (e.g., md5sum), adherence to a data specification (e.g., JSON Schema), and data semantics (e.g., SHACL for linked data).

Finally, metadata (automatic versus manual annotation) should not be neglected. Automatic generation of metadata might require predefined standards and an ontology to fully track how the data was generated [22].

Conclusion

Benchmarking is a cornerstone practice in bioinformatics, impacting how methods are conceived, developed, published and chosen. Benchmarking is driven by the effort of people with the expertise to address biological and technical problems. This expertise can be leveraged by current practices of community benchmarking, so scientific advances are eased by joint efforts to catalog open problems, generate (or simulate) appropriate input data, develop methods to address them, and evaluate their performances. Community benchmarking has been proven successful and productive [13,19,21].

Here, we discussed the motivation and process of community benchmarking at several levels, from definition to execution, while highlighting strategies and technologies to potentially ease the technical and scientific challenges as well as how to organize contributions from communities.

Acknowledgments

We thank the Robinsonlab members for their constructive feedback.

Author contributions

Conceptualization: Izaskun Mallona, Charlotte Soneson, Ben Carrillo, Almut Lütge, Daniel Incicau, Reto Gerber, Anthony Sonrel, Mark D. Robinson.

Formal analysis: Izaskun Mallona.

Investigation: Izaskun Mallona.

Supervision: Izaskun Mallona, Mark D. Robinson.

Validation: Izaskun Mallona.

Writing – original draft: Izaskun Mallona, Charlotte Soneson, Ben Carrillo, Almut Lütge, Daniel Incicau, Reto Gerber, Anthony Sonrel, Mark D. Robinson.

Writing – review & editing: Izaskun Mallona, Charlotte Soneson, Ben Carrillo, Almut Lütge, Daniel Incicau, Reto Gerber, Anthony Sonrel, Mark D. Robinson.

References

1. Cao Y, Yu L, Torkel M, Kim S, Lin Y, Yang P, et al. The current landscape and emerging challenges of benchmarking single-cell methods. Brief Bioinform. 2025;26(5):bbaf380. https://doi.org/10.1093/bib/bbaf380 PMID: 41045508
2. Sonrel A, Luetge A, Soneson C, Mallona I, Germain P-L, Knyazev S, et al. Meta-analysis of (single-cell method) benchmarks reveals the need for extensibility and interoperability. Genome Biol. 2023;24(1):119. https://doi.org/10.1186/s13059-023-02962-5 PMID: 37198712
3. Boulesteix A-L, Lauer S, Eugster MJA. A plea for neutral comparison studies in computational sciences. PLoS One. 2013;8(4):e61562. https://doi.org/10.1371/journal.pone.0061562 PMID: 23637855
4. Weber LM, Saelens W, Cannoodt R, Soneson C, Hapfelmeier A, Gardner PP, et al. Essential guidelines for computational method benchmarking. Genome Biol. 2019;20(1):125. https://doi.org/10.1186/s13059-019-1738-8 PMID: 31221194
5. Brooks TG, Lahens NF, Mrčela A, Grant GR. Challenges and best practices in omics benchmarking. Nat Rev Genet. 2024;25(5):326–39. https://doi.org/10.1038/s41576-023-00679-6 PMID: 38216661
6. Rich JM, Moses L, Einarsson PH, Jackson K, Luebbert L, Booeshaghi AS. The impact of package selection and versioning on single-cell RNA-seq analysis. bioRxiv. 2024;:2024.04.04.588111. https://doi.org/10.1101/2024.04.04.588111 PMID: 38617255
7. Wilkinson MD, Dumontier M, Aalbersberg IJJ, Appleton G, Axton M, Baak A, et al. The FAIR guiding principles for scientific data management and stewardship. Sci Data. 2016;3:160018. https://doi.org/10.1038/sdata.2016.18 PMID: 26978244
8. Nevers Y, Jones TEM, Jyothi D, Yates B, Ferret M, Portell-Silva L, et al. The quest for orthologs orthology benchmark service in 2022. Nucleic Acids Res. 2022;50(W1):W623–32. https://doi.org/10.1093/nar/gkac330 PMID: 35552456
9. Bryce-Smith S, Burri D, Gazzara MR, Herrmann CJ, Danecka W, Fitzsimmons CM, et al. Extensible benchmarking of methods that identify and quantify polyadenylation sites from RNA-seq data. RNA. 2023;29(12):1839–55. https://doi.org/10.1261/rna.079849.123 PMID: 37816550
10. Pardo-Palacios FJ, Wang D, Reese F, Diekhans M, Carbonell-Sala S, Williams B, et al. Systematic assessment of long-read RNA-seq methods for transcript identification and quantification. Nat Methods. 2024;21(7):1349–63. https://doi.org/10.1038/s41592-024-02298-3 PMID: 38849569
11. Meyer P, Saez-Rodriguez J. Advances in systems biology modeling: 10 years of crowdsourcing DREAM challenges. Cell Syst. 2021;12(6):636–53. https://doi.org/10.1016/j.cels.2021.05.015 PMID: 34139170
12. Zook JM, McDaniel J, Olson ND, Wagner J, Parikh H, Heaton H, et al. An open resource for accurately benchmarking small variant and reference calls. Nat Biotechnol. 2019;37(5):561–6. https://doi.org/10.1038/s41587-019-0074-6 PMID: 30936564
13. Moult J. A decade of CASP: progress, bottlenecks and prognosis in protein structure prediction. Curr Opin Struct Biol. 2005;15(3):285–9. https://doi.org/10.1016/j.sbi.2005.05.011 PMID: 15939584
14. Wratten L, Wilm A, Göke J. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. Nat Methods. 2021;18(10):1161–8. https://doi.org/10.1038/s41592-021-01254-9 PMID: 34556866
15. Amstutz P, Mikheev M, Crusoe MR, Tijanic N, Lampa S. Existing workflow systems. 2024. https://s.apache.org/existing-workflow-systems
16. Amstutz P, Crusoe MR, Tijanić N, Chapman B, Chilton J, Heuer M, et al. Common Workflow Language, v1.0. 2016. https://research.manchester.ac.uk/files/57032693/cwl_1.0_workflow.pdf

17. Goble C, Cohen-Boulakia S, Soiland-Reyes S, Garijo D, Gil Y, Crusoe MR, et al. FAIR computational workflows. Data Intellegence. 2020;2(1–2):108–21. https://doi.org/10.1162/dint_a_00033
18. Hanssen F, Gabernet G, Bäuerle F, Stöcker B, Wiegand F, Smith NH, et al. NCBench: providing an open, reproducible, transparent, adaptable, and continuous benchmark approach for DNA-sequencing-based variant calling. F1000Res. 2024;12:1125. https://doi.org/10.12688/f1000research.140344.2 PMID: 39345270
19. Capella-Gutierrez S, de la Iglesia D, Haas J, Lourenco A, Fernández JM, Repchevsky D. Lessons learned: recommendations for establishing critical periodic scientific benchmarking. bioRxiv. 2017:181677. https://doi.org/10.1101/181677
20. Cannoodt R, Cannoodt H, Schaumont D, Waldrant K, Van de Kerckhove E, Boschmans A, et al. Viash: a meta-framework for building reusable workflow modules. JOSS. 2024;9(93):6089. https://doi.org/10.21105/joss.06089
21. Luecken MD, Gigante S, Burkhardt DB, Cannoodt R, Strobl DC, Markov NS, et al. Defining and benchmarking open problems in single-cell analysis. Nat Biotechnol. 2025;43(7):1035–40. https://doi.org/10.1038/s41587-025-02694-w PMID: 40595413
22. Lebo T, Sahoo S, McGuinness D, Belhajjame K, Cheney J, Corsar D. PROV-O: The PROV ontology. 2013. http://www.w3.org/TR/2013/REC-prov-o-20130430/
23. Bechhofer S, Buchan I, De Roure D, Missier P, Ainsworth J, Bhagat J, et al. Why linked data is not enough for scientists. Future Generation Computer Systems. 2013;29(2):599–611. https://doi.org/10.1016/j.future.2011.08.004
24. Soiland-Reyes S, Sefton P, Crosas M, Castro LJ, Coppens F, Fernandez JM, et al. Packaging research artefacts with RO-Crate. Data Science. 2022;5(2):97–138. https://doi.org/10.3233/DS-210053
25. Khan RU, Zhang X, Kumar R, Aboagye EO. Evaluating the performance of ResNet model based on image recognition. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence. 2018. p. 86–90. https://doi.org/10.1145/3194452.3194461
26. Jumper J, Evans R, Pritzel A, Green T, Figurnov M, Ronneberger O, et al. Highly accurate protein structure prediction with AlphaFold. Nature. 2021;596(7873):583–9. https://doi.org/10.1038/s41586-021-03819-2 PMID: 34265844
27. Lütge A, Zyprych-Walczak J, Brykczynska Kunzmann U, Crowell HL, Calini D, Malhotra D, et al. CellMixS: quantifying and visualizing batch effects in single-cell RNA-seq data. Life Sci Alliance. 2021;4(6):e202001004. https://doi.org/10.26508/lsa.202001004 PMID: 33758076
28. Reinke A, Tizabi MD, Baumgartner M, Eisenmann M, Heckmann-Nötzel D, Kavur AE, et al. Understanding metric-related pitfalls in image analysis validation. Nat Methods. 2024;21(2):182–94. https://doi.org/10.1038/s41592-023-02150-0 PMID: 38347140
29. Cannoodt R, Saelens W. https://cran.r-project.org/web/packages/funkyheatmap/index.html
30. Saelens W, Cannoodt R, Todorov H, Saeys Y. A comparison of single-cell trajectory inference methods. Nat Biotechnol. 2019;37(5):547–54. https://doi.org/10.1038/s41587-019-0071-9 PMID: 30936559
31. Jelizarow M, Guillemot V, Tenenhaus A, Strimmer K, Boulesteix A-L. Over-optimism in bioinformatics: an illustration. Bioinformatics. 2010;26(16):1990–8. https://doi.org/10.1093/bioinformatics/btq323 PMID: 20581402
32. Boulesteix A-L. Over-optimism in bioinformatics research. Bioinformatics. 2010;26(3):437–9. https://doi.org/10.1093/bioinformatics/btp648 PMID: 19942585
33. Strobl C, Leisch F. Against the “one method fits all data sets” philosophy for comparison studies in methodological research. Biom J. 2024;66(1):e2200104. https://doi.org/10.1002/bimj.202200104 PMID: 36053253
34. Marini F, Soneson C. 2024. https://bioconductor.org/packages/bettr/
35. Taherdoost H, Madanchian M. Multi-Criteria Decision Making (MCDM) methods and concepts. Encyclopedia. 2023;3(1):77–87. https://doi.org/10.3390/encyclopedia3010006
36. Nießl C, Herrmann M, Wiedemann C, Casalicchio G, Boulesteix AL. Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. Wiley Interdiscip Rev Data Min Knowl Discov. 2022;12:e1441.
37. Mallona I, Luetge A, Carrillo B, Incicau D, Gerber R, Sonrel A, et al. Omnibenchmark (alpha) for continuous and open benchmarking in bioinformatics. arXiv preprint 2024. https://arxiv.org/abs/2409.17038
38. Hatos A, Quaglia F, Piovesan D, Tosatto SCE. APICURON: a database to credit and acknowledge the work of biocurators. Database (Oxford). 2021;2021:baab019. https://doi.org/10.1093/database/baab019 PMID: 33882120
39. Druskat S, Spaaks JH, Chue Hong N, Haines R, Baker J, Bliven S, et al. Citation file format; 2021. https://doi.org/10.5281/zenodo.5171937
40. Greenstein S, Zhu F. Open content, Linus’ law, and neutral point of view. Inf Syst Res. 2016;27:618–35. https://doi.org/10.1287/isre.2016.0643
41. Sullivan I, DeHaven A, Mellor D. Open and reproducible research on open science framework. Curr Protoc Essent Lab Tech. 2019;18:e32. https://doi.org/10.1002/cpet.32
42. Olevska A, Bert B, Ebrahimi L, Schonfelder G, Heinl C. Ensuring reproducible research requires a support infrastructure: the value of public registries to publishers. Sci Editor. 2021. p. 4–7. https://doi.org/10.36591/SE-D-4401-4
43. Teo AYY, Squair JW, Courtine G, Skinnider MA. Best practices for differential accessibility analysis in single-cell epigenomics. Nat Commun. 2024;15(1):8805. https://doi.org/10.1038/s41467-024-53089-5 PMID: 39394227
44. Siepe BS, Bartoš F, Morris TP, Boulesteix AL, Heck DW, Pawel S. Simulation studies for methodological research in psychology: a standardized template for planning, preregistration, and reporting. Psychol Methods. 2024. https://doi.org/10.1037/met0000695 PMID 39541533
45. Simpkin AJ, Mesdaghi S, Sánchez Rodríguez F, Elliott L, Murphy DL, Kryshtafovych A, et al. Tertiary structure assessment at CASP15. Proteins. 2023;91(12):1616–35. https://doi.org/10.1002/prot.26593 PMID: 37746927

46. Kryshtafovych A, Schwede T, Topf M, Fidelis K, Moult J. Critical assessment of methods of protein structure prediction (CASP)-Round XIII. Proteins. 2019;87(12):1011–20. https://doi.org/10.1002/prot.25823 PMID: 31589781
47. Howison J, Deelman E, McLennan MJ, Ferreira da Silva R, Herbsleb JD. Understanding the scientific software ecosystem and its impact: current and future measures. Res Eval. 2015;24:454–70. https://doi.org/10.1093/reseval/rvv014
48. Trisovic A, Lau MK, Pasquier T, Crosas M. A large-scale study on research code quality and execution. Sci Data. 2022;9(1):60. https://doi.org/10.1038/s41597-022-01143-6 PMID: 35190569
49. Pike R, Kernighan BW. The UNIX System: program design in the UNIX Environment. AT&T Bell Laboratories Technical Journal. 1984;63(8):1595–605. https://doi.org/10.1002/j.1538-7305.1984.tb00055.x
50. Proctor RW, Schneider DW. Hick's law for choice reaction time: a review. Q J Exp Psychol (Hove). 2018;71(6):1281–99. https://doi.org/10.1080/17470218.2017.1322622 PMID: 28434379
51. Procida D. Diátaxis, a systematic approach to technical documentation authoring. https://diataxis.fr/
52. Hill D, Antunes BA, Bertrand A, Nguifo EM, Yon L, Nautré-Domanski J. Machine learning and reproducibility impact of random numbers. 2024. https://hal.science/hal-04642175v1
53. Dröge B, Holanda Rusu V, Hoste K, van Leeuwen C, O'Cais A, Röblitz T. EESSI: a cross-platform ready-to-use optimised scientific software stack. Software: Practice and Experience. 2023;53:176–210. https://doi.org/10.1002/spe.3075
54. Hoste K, Timmerman J, Georges A, De Weirdt S. EasyBuild: building software with ease. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. 2012. p. 572–82. https://ieeexplore.ieee.org/abstract/document/6495863
55. Gamblin T, LeGendre M, Collette MR, Lee GL, Moody A, de Supinski BR, et al. The Spack package manager. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015. p. 1–12. https://doi.org/10.1145/2807591.2807623
56. Kurtzer GM, Sochat V, Bauer MW. Singularity: scientific containers for mobility of compute. PLoS One. 2017;12(5):e0177459. https://doi.org/10.1371/journal.pone.0177459 PMID: 28494014
57. Blomer J, Buncic P, Meusel R. The CernVM file system. Geneva, Switzerland: CERN; 2013.
58. Boissonneault M, Oldeman BE, Taylor RP. Providing a unified software environment for Canada's National Advanced Computing Centers. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning). 2019. p. 1–6. https://doi.org/10.1145/3332186.3332210
59. Laine C, Goodman SN, Griswold ME, Sox HC. Reproducible research: moving toward research the public can really trust. Ann Intern Med. 2007;146:450. https://doi.org/10.7326/0003-4819-146-6-200703200-00154 PMID 17339612
60. Kreutzer T. Open content: a practical guide to using creative commons licences; 2014. https://meta.wikimedia.org/wiki/Open_Content_-_A_Practical_Guide_to_Using_Creative_Commons_Licences/The_basics_of_Open_Content_licencing/en
61. Potter M, Smith T. Making code citable with Zenodo and GitHub. 2015. https://zenodo.org/record/45042
62. Sicilia M-A, García-Barriocanal E, Sánchez-Alonso S. Community curation in open dataset repositories: insights from Zenodo. Procedia Computer Science. 2017;106:54–60. https://doi.org/10.1016/j.procs.2017.03.009
63. van de Sandt S, Nielsen LH, Ioannidis A, Muench A, Henneken E, Accomazzi A. Practice meets principle: tracking software and data citations to Zenodo DOIs. arXiv preprint 2019. https://arxiv.org/abs/1911.00295
64. Lamb C, Zacchiroli S. Reproducible builds: increasing the integrity of software supply chains. arXiv preprint 2021. https://arxiv.org/abs/2104.06020
65. Huynh D. How Python Data Science Libraries Can Be Hijacked (and What You Can Do About It). 2023. https://blog.mithrilsecurity.io/how-python-data-science-libraries-can-be-hijacked-and-what-you-can-do-about-it-2/
66. Bleih A. New vulnerability in R's deserialization discovered. https://cyberint.com/blog/research/new-vulnerability-in-rs-deserialization-discovered/