Research software as scientific evidence

Toward shared specifications for research software disclosure

Introduction

Research software is pivotal in scientific research, generating, processing, and analyzing results intended for publications (*1*). Its increasing integration into academic activities is a testament to its significance across the scientific community, prompting a greater focus on its development practices. Many efforts have emerged to strengthen accessibility, replicability, transparency, and reusability in research software by establishing guidelines, recommending good practices, developing verification platforms, and offering educational resources (*2*-*5*). Yet these efforts tend to emphasize the *software* aspect, without clarifying what additional role is implied by the *research* aspect.

We refer to this additional role as the evidentiary role — the idea that research software should serve as part of the evidence that supports scientific claims, beyond its technical function. This evidentiary role shifts the focus away from how to develop research software toward the question of how it should be disclosed and under what specifications, specifically what constitutes evidentiary sufficiency in the context of a given scientific claim. However, no shared specifications currently exist within the stakeholder community (including authors, editors, and reviewers) on how research software should be disclosed to support the evidentiary role.

This lack of consensus has produced concrete consequences, with a notable case being *Nature*'s decision to publish the AlphaFold3 paper without its code (*6*). As required in *Nature Portfolio Editorial Policies*, code supporting central claims is to be made available to reviewers upon request. Despite the stated policies, *Science* reported that one reviewer had only temporary access to an early web server and described repeated, unanswered requests for code. Later, *Nature* cited the biosecurity risks and the inclusion of pseudocode to justify the editorial decision, while *DeepMind* attributed the restriction to commercial considerations. This case illustrates how each stakeholder operates from a distinct position and how the absence of shared expectations can lead to tensions in disclosure decisions.

Beyond tensions, the more serious concern is the gray zone created by this lack of consensus. In this zone, disclosure practices are inconsistently interpreted and selectively applied. Researchers acting in good faith may believe that, as long as they have subjectively avoided data fabrication and result concealment, there is no need to consider what constitutes sufficient disclosure. Meanwhile, mounting evidence shows how others have exploited this lack of consensus to justify withholding critical software components or data elements. In both situations, evidentiary insufficiency can persist without shared disclosure specifications. This goes beyond what case-by-case discretion can fully resolve.

Why, then, is there still no clear consensus on disclosure specifications? One reason lies in the wide variability of disclosure norms across scientific fields. In some fields, community standards guide software sharing and verification, while disclosure remains informal or absent in others. A second complicating factor is the influence of external disclosure barriers such as proprietary licensing, commercial interests, and biosecurity concerns, all of which introduce competing priorities that shape what can or should be disclosed. Together, these factors constrain the broader establishment of shared disclosure specifications, making it increasingly difficult to keep pace with the growing interdisciplinarity and complexity of contemporary research practice.

Herein, we report the outcome of preliminary discussions involving representatives from the stakeholder community that helped initiate the preliminary development of shared disclosure specifications. Building on insights from these discussions, we propose a new conceptual model called Evidence-Oriented Programming. Rather than focusing on how research software is efficiently implemented, this model reorients software development and evaluation around the relationship between published scientific claims and computational artifacts (i.e., software components and data elements) that substantiate them. To operationalize this model, we introduce the Evidence Chain Method — a strategy for identifying and linking the evidentiary computational artifacts. We discuss its applicability across different fields and propose how it may help resolve trust gaps when external disclosure barriers arise. Overall, these efforts seek to support a shift from case-by-case disclosure decisions toward evidence-oriented specifications across the scientific ecosystem.

From Replicability to Evidentiary Sufficiency

Replication has long been central to how scientists establish confidence in the scientific merit of results, supported by a shared body of community positions and guidance frameworks, such as the FAIR4RS Principles (*2*), the TOP Guidelines (*7*), the OpenAIRE Guidelines, and the COAR Framework (*8*).Thus, it is worth asking why evidentiary sufficiency, rather than replicability, is our primary concern.

- Replicability is not universally attainable. It is substantively undermined by external disclosure barriers and reliance on proprietary software or hardware environments, which are common realities in contemporary research ecosystems.
- The mere success or failure of replication is an inadequate indicator of the health of science (*9*). A successful replication may simply reproduce unrecognized systematic errors (*10*). Conversely, a failed replication does not necessarily refute the original claims, as such outcomes might arise from undocumented parameters, uncontrolled variables, or even stochastic effects.
- Providing research software does not guarantee reproducibility. A recent reproducibility analysis of over 2,000 R code repositories found that only 26% of scripts could be successfully executed on the first attempt, and even after code cleaning, the success rate reached only 44% (*11*).

These realities highlight the need to assess scientific validity in the context of the full evidentiary structure, rather than focusing on isolated acts of replication, and they motivate our exploration of how research software can help construct and support such a context.

Evidence-Oriented Programming

While evidentiary sufficiency can, in principle, be assessed at any stage of research, the peer review stage represents the most critical juncture. Only during peer review, when publication decisions are still pending, does sufficient leverage exist to address potential evidentiary insufficiency. Once publication has occurred, it becomes significantly more challenging to request additional disclosure, and any errors that surface may be considerably harder to correct in a timely manner, potentially allowing flawed results to propagate more widely before correction can occur (*10*).

For the specific context of the peer review stage, we have developed a conceptual model, called Evidence-Oriented Programming (EOP), to examine the tensions between stakeholder priorities and the possibilities for negotiated consensus. Any attempt to assess evidentiary sufficiency during peer review must begin with stakeholder consensus on the scope, timing, and form of disclosure.

1. **Scope.** Full openness remains a desirable ideal, but it is not always feasible or appropriate, as different stakeholder groups — authors, editors, reviewers, and external actors (such as regulatory agencies, funding bodies, and other entities that exert oversight or guidance over disclosure practices) — bring distinct and competing priorities. As illustrated in Figure 1A, the final disclosure state of research software often reflects the interplay of these priorities rather than adherence to a single normative standard. EOP does not seek to persuade all stakeholders to adopt full openness. Instead, it accepts the reality of competing priorities and aims to balance legitimate constraints with the evidentiary integrity needed to support scientific claims and reduce the risk of serious downstream consequences (Figure 1B). While the minimum bounds of acceptable disclosure remain unclear, it should exclude practices collectively regarded as unacceptable by the stakeholder community. Potentially, retraction cases can serve as concrete reference points for identifying unacceptable disclosure practices.

2. **Timing.** In most cases, authors can make research software fully accessible at the time of submission, and we encourage them to do so. However, potential conflicts of interest between authors and reviewers must not be overlooked, particularly when reviewers might improperly use the research software to publish overlapping work ahead of the authors (*12*). In such cases, EOP needs to support authors in providing a mandatory disclosure schedule, explicitly indicating which components remain withheld, including associated hash values to enable verification when these components are eventually disclosed, along with a justification of the disclosure risks and their relevance to the scientific claims under review.
3. **Form.** Evaluating research software poses challenges distinct from general-purpose software review (*13*). Firstly, research software often includes novel data types, modeling assumptions, or uncommon dependencies. While they also occur in general-purpose software, they become especially consequential in research settings, where their coupling to the evidentiary evaluation of scientific claims can render evaluation during peer review difficult or even infeasible. In addition, external disclosure barriers may prevent full source code availability, with critical components accessible only through compiled binaries or restricted endpoints, sometimes under time-limited conditions. Furthermore, reviewers may lack the domain expertise required to evaluate complex or poorly documented research software. Finally, single- and double-blind review formats restrict author-reviewer interaction, reducing opportunities to resolve ambiguities or negotiate appropriate disclosure standards. Accordingly, EOP must encompass a structured approach to representing research software, helping stakeholders align on how it should be disclosed, interpreted, and evaluated.

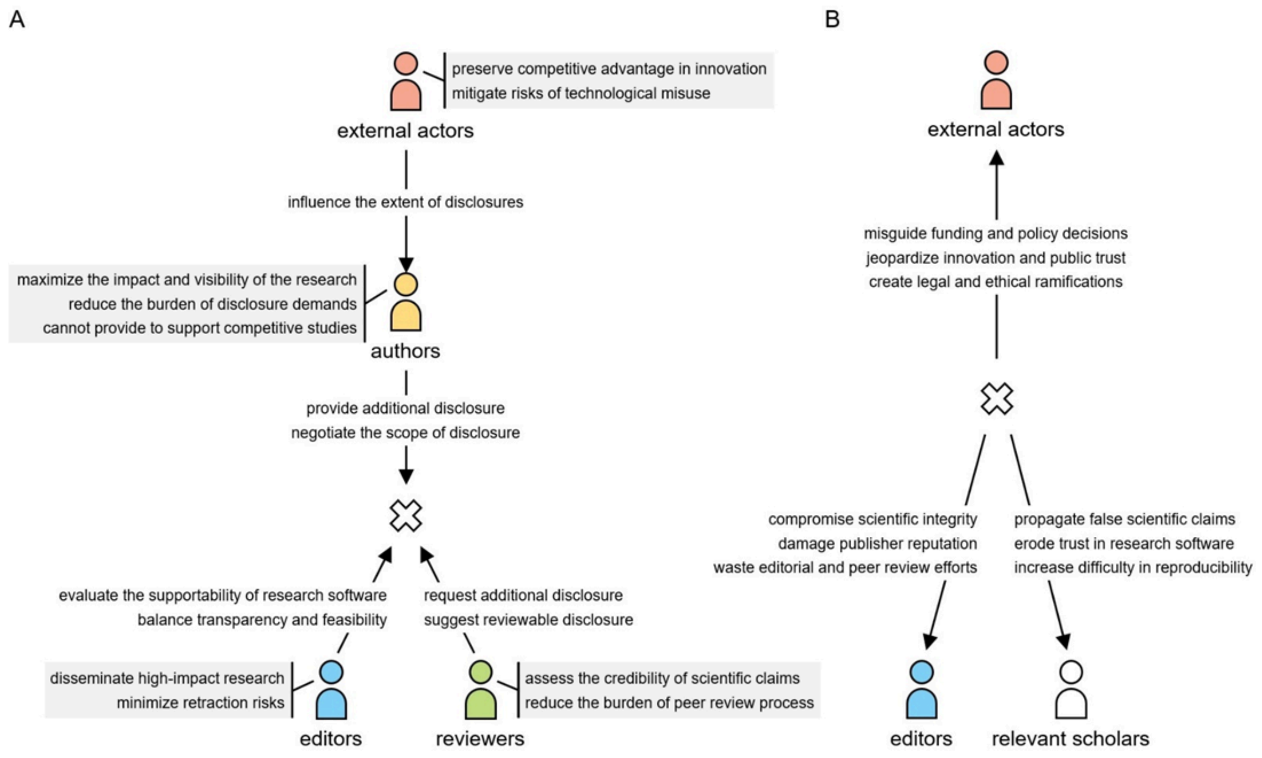

A
preserve competitive advantage in innovation
mitigate risks of technological misuse
external actors
influence the extent of disclosures
maximize the impact and visibility of the research
reduce the burden of disclosure demands
cannot provide to support competitive studies
authors
provide additional disclosure
negotiate the scope of disclosure
evaluate the supportability of research software
balance transparency and feasibility
request additional disclosure
suggest reviewable disclosure
disseminate high-impact research
minimize retraction risks
editors
reviewers
assess the credibility of scientific claims
reduce the burden of peer review process
B
external actors
misguide funding and policy decisions
jeopardize innovation and public trust
create legal and ethical ramifications
compromise scientific integrity
damage publisher reputation
waste editorial and peer review efforts
propagate false scientific claims
erode trust in research software
increase difficulty in reproducibility
editors
relevant scholars

Figure 1 | Interactions within the stakeholder community. (A) The priorities and actions of four stakeholder groups forming the tension. (B) The serious downstream consequences from this structural ambiguity within the stakeholder community.

Evidence Chain Method

The Evidence Chain Method (ECM), which we developed by adapting ideas from scientific workflows (*14*), offers a concrete way to put EOP into practice. One of the core difficulties it addresses is the invisibility of evidentiary relationships: individual software modules or datasets often lack the necessary context to clarify their role in supporting a scientific claim. ECM responds to this challenge by mapping computational artifacts to the claim they support, making evidentiary structures explicit, interpretable, and available for review.

ECM introduces a bounded structure that focuses the assessment of evidentiary sufficiency between two key points: the starting and ending artifacts. The starting artifacts are the *input data* explicitly identified in the publication, such as publicly available datasets, experimentally collected measurements, or algorithmically generated values. The ending artifacts are the resulting *visual claims*: figures, tables, or reported statistical values such as correlation coefficients or confidence levels that readers interpret as evidence in the publication. This range makes clear that ECM is designed to ensure that, once input data are declared and used in software-mediated processes, the resulting claims can be traced and evaluated for evidentiary sufficiency. The compliance of non-software-mediated procedures and the integrity of their data collection processes fall beyond the scope of ECM.

To link the starting and ending artifacts, ECM provides an abstracted evidence chain (Figure 2A). First, *input data* are transformed into *output data* through the *experimental or analytical process(es)*. These *output data* can include processed datasets, trained model archives, or other intermediate data elements. A subset of the *output data* constitutes the *visual data*, which are then converted via the *plotting or summarizing process(es)* to produce the *visual claims*. Equally important is the *documentation* maintained across these artifacts, which explains both the overarching structure and specific linkages, ensuring that the trace remains coherent and interpretable.

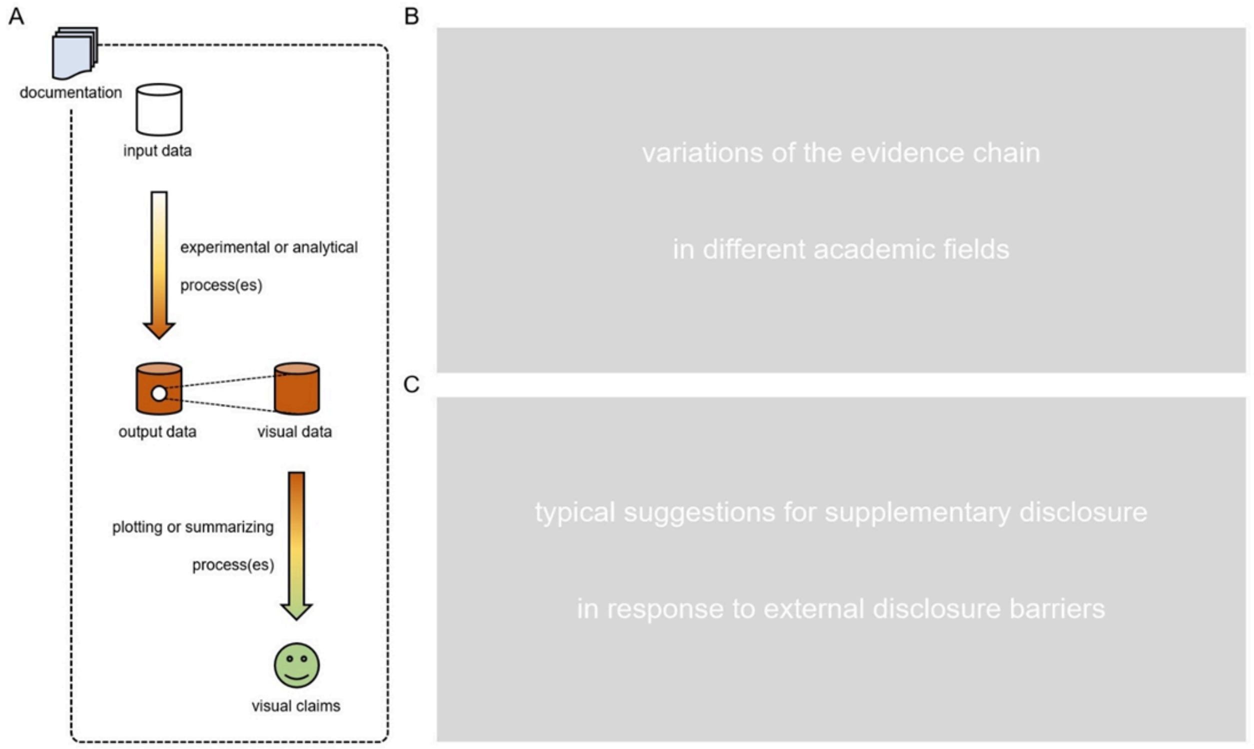

A
documentation
input data
experimental or analytical
process(es)
output data
visual data
plotting or summarizing
process(es)
visual claims
B
variations of the evidence chain
in different academic fields
C
typical suggestions for supplementary disclosure
in response to external disclosure barriers

Figure 2 | Evidence chain method and its variations. (A) Structure of the basic evidence chain. (B) Variations of the evidence chain in different academic fields. (C) Typical suggestions for supplementary disclosure in response to external disclosure barriers. A detailed discussion of (B) and (C) can be found in the Supplementary Information.

While these seven artifacts form a basic evidence chain, different academic fields can introduce additional complexity, especially within the *experimental or analytical process(es)*. Depending on whether the scientific claims are algorithm-centered, this artifact may range from simple software invocations to sophisticated pipelines that incorporate the authors' proposed algorithms alongside baseline or comparator algorithms. Further, additional data elements such as hyperparameter configurations, random seeds, and initialization routines also become critical to evidentiary interpretation. These differences are not merely technical; they reflect distinct evidentiary norms and disclosure expectations across fields. ECM can accommodate such variation by adapting to the distinctive characteristics of different academic fields, enabling scientific claims to remain evidentially sound even as fields diverge. Field-specific adaptations of ECM are further elaborated in the Supplementary Information.

In practice, full disclosure of evidentiary software components and data elements may not be feasible due to ethical, legal, security, or intellectual property constraints. ECM allows trade-off presentations, such as providing partial outputs, operational logs, intermediate output data with evidential significance, and/or typical case presentations that preserve evidentiary traceability without unrestricted access to all underlying components. These mechanisms help reviewers and editors assess scientific claims when direct disclosure is constrained. Representative strategies and their evidentiary rationale are discussed in Supplementary Information.

Overall, the basic evidence chain, field-specific variants, and trade-off presentations in ECM collectively instantiate the *scope* and *form* dimensions of EOP. To extend ECM toward the *timing* dimension, one can simulate selective non-disclosure by omitting or withholding a given component to evaluate its precise impact on evidentiary sufficiency. This enables the stakeholder community to assess whether, and how, delayed or withheld disclosure compromises the ability to substantiate scientific claims.

Education and Evaluation

Although ECM offers a concrete operationalization of EOP, it presents many practical challenges. On the implementation side, ECM places substantial demands on authors. It requires proficiency in software design and documentation practices, as well as the ability to present them in a form that supports evaluation. These are skills that many researchers, especially those without formal training in software engineering, may lack. On the evaluation side, ECM remains an early-stage method. Its field-specific variants have yet to be systematically developed, and no large-scale quantitative studies have been conducted to assess its applicability or evidentiary effectiveness across academic fields.

To reduce the practical barriers to the implementation of ECM, we are developing educational tools that reflect how researchers actually engage with software in their academic fields. In addition to static tutorials, these resources include layered guides and workshop-style modules built around case studies, each designed to foreground the evidentiary chains most relevant to a given field. We are also testing ways to embed ECM into existing scientific workflows and exploring whether AI agents might help researchers identify and annotate relevant artifacts during software development.

In parallel, we are working with software quality assurance communities and academic publishers to build the structural conditions necessary for ECM to take root. This involves developing editorial guidelines for reviewers and editors, and partnering with platforms such as Code Ocean to prototype features that support evidence chain visibility. These efforts aim not just to make ECM technically feasible, but to embed evidentiary review into the publishing process in ways that are practical and scalable.

Conditional Full Openness

While EOP aims to address routine disclosure practices, we recognize that exceptional situations might warrant more decisive measures. We suggest that, prior to manuscript acceptance, authors predefine which designated audiences (e.g., regulatory bodies, ethics boards, or expert oversight panels) would be granted full access to their research software under specified conditions — namely, when such work attracts widespread attention and becomes the subject of significant technical doubts after publication. Although this proposal extends beyond the technical scope of Evidence-Oriented Programming itself, it touches on related concerns about evidentiary accountability and might warrant consideration in the development of disclosure policies.

Outlook

This work reflects a concerted effort by members of the stakeholder community to address the evidentiary ambiguity that persists across research software practices. It seeks to clarify the gray zone created by the absence of consensus on disclosure, where disclosure practices are inconsistently interpreted and selectively applied, and to offer principles for mitigating its effects. The introduction of EOP and ECM is not intended to replace earlier replicability-focused efforts, but to complement them by acknowledging the real-world constraints and the need for claim-centered, disclosure-aware evaluation.

We hope this work will serve as a catalyst for broader engagement across the stakeholder community in developing shared expectations around evidentiary sufficiency. Supported by incentives for structured software disclosures, such expectations can foster a research ecosystem in which software is not merely a computational tool but a verifiable contributor to scientific knowledge. Achieving this vision will require not only technical and institutional readiness, but also a shared commitment to treating research software as a site of evidentiary accountability — and acting on that commitment.

References and Notes

1. M. Gruenpeter et al. Defining research software: a controversial discussion. *Zenodo* (2021).
2. M. Barker et al. Introducing the FAIR principles for research software. *Scientific Data* **9** (2022), 622.
3. H. Hunter-Zinck et al. Ten simple rules on writing clean and reliable open-source scientific software. *PLoS Computational Biology* **17** (2021), e1009481.
4. T. Staubitz et al. CodeOcean – A versatile platform for practical programming exercises in online environments. *IEEE Global Engineering Education Conference* (2016), 314–323.
5. G. Wilson. Software carpentry: getting scientists to write better code by making them more productive. *Computing in Science & Engineering* **8** (2006), 66–69.
6. S. Wankowicz et al. AlphaFold3 Transparency and Reproducibility. *Zenodo* (2024).
7. B. A. Nosek et al. Promoting an open research culture. *Science* **348** (2015), 1422–1425.
8. Confederation of Open Access Repositories. COAR Community Framework for Good Practices in Repositories. *Zenodo* (2022).
9. National Academies of Sciences, Engineering, and Medicine, et al. Reproducibility and Replicability in Science (2019) *National Academies Press*
10. G. Miller. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science* **314** (2006), 1856–1857.
11. A. Trisovic, et al. A large-scale study on research code quality and execution. *Scientific Data* **9** (2022), 1–16.
12. https://ori.hhs.gov/case-three-getting-scooped-reviewer
13. W. W. Royce. Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th International Conference on Software Engineering* (1987), 328–338.
14. C. S. Liew et al. Scientific workflows: moving across paradigms. *ACM Computing Surveys* **49** (2016), 1–39.

~~Introduction~~

~~Research software is pivotal in scientific research, generating, processing, and analyzing results intended for publications (1). Its increasing integration into academic activities acknowledges its significance across the scientific community. This increased recognition has led to growing expectations for research software, prompting a greater focus on its development practices. As a consequence, well-adopted efforts like Findable, Accessible, Interoperable, and Reusable for Research Software (FAIR4RS) guiding principles have been crucial to enhancing the quality of research software and aligning it with the characteristics required for effective interaction within the scientific community (2).~~

~~However, these efforts do not seem to prevent retraction incidents related to research software effectively. Many of such cases originate from reliance on fabricated datasets, flawed statistical methodologies, or selective reporting, all of which can be exacerbated and more challenging to detect due to inadequate disclosure of research software (3). Without transparent access to code and data, research findings – particularly those with striking scientific claims – can bypass scrutiny during peer review and achieve publication in leading journals, only to be discredited later (4-8). As famously stated by Carl Sagan, "extraordinary claims require extraordinary evidence". It has become increasingly clear, with the rise of computational techniques in science, that research software is a component in a supplementary evidence.~~

~~The recent controversy surrounding *Nature* regarding the release timing of the AlphaFold3 paper (9) further puts the shortcomings in existing efforts of research software on presentation. The absence of explicit specifications for determining which components of research software are essential and when they must be publicly disclosed. This ambiguity becomes particularly problematic when research software is influenced by external disclosure barriers such as commercial pressures or concerns over security and sensitivity. In such cases, adjustments of whether the research software meets expectations for openness, even when contextually reasonable, risk appearing highly subjective. This lack of clarity might inadvertently provide an opportunity for academic misconduct.~~

~~Overall, while existing efforts are well intended, they suffer from limitations in their applicability as specifications for disclosing research software in the peer review processes. To this end, we advocate for an objective balance between the credibility of research software and the external disclosure barriers it has, emphasizing the need for consensus on actionable specifications to complement existing efforts. To this end, we aim to facilitate extensive discussions among editors, reviewers, authors, and external forces such as funding agencies and regulatory bodies, proposing a new programming paradigm (10) that we term Evidence-Oriented Programming.~~

~~Multi-Stakeholder Conflicting Priorities on Disclosure~~

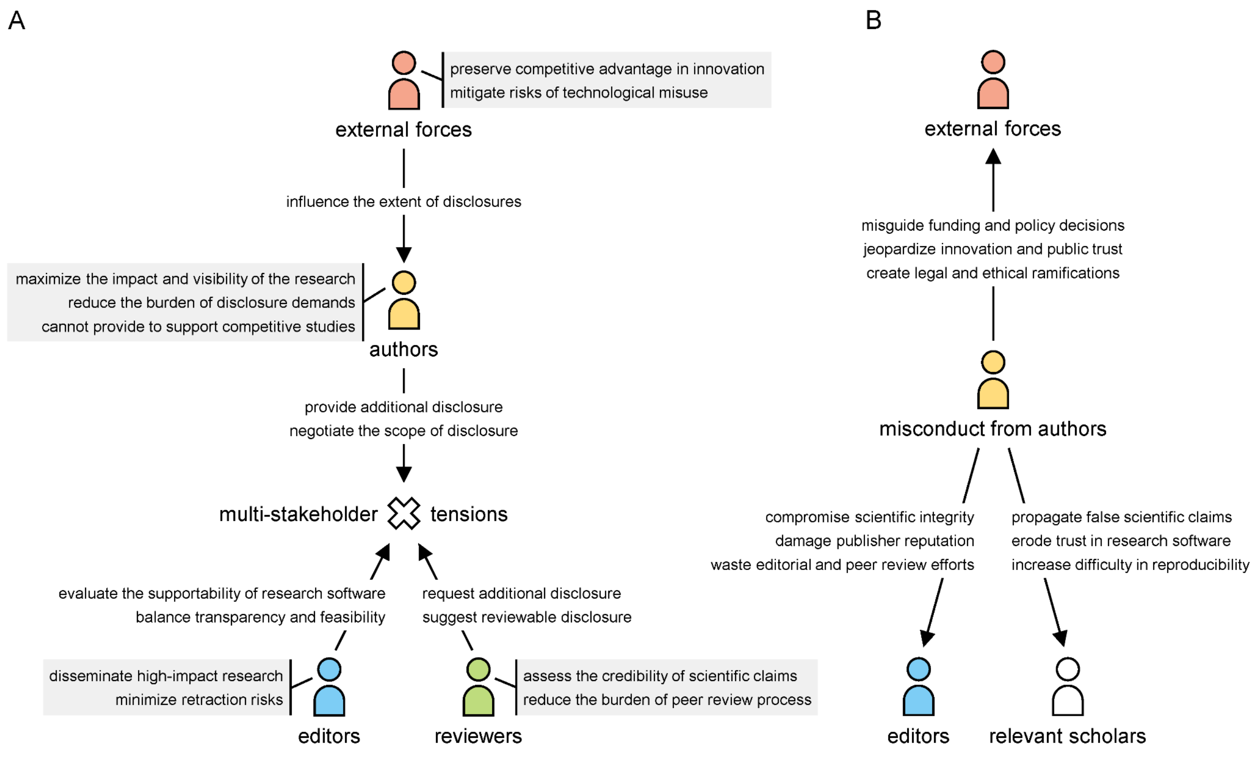
~~Defining actionable specifications for research software disclosure is complicated by the distinct priorities of different stakeholders. These priorities create inherent tensions in the peer review process, making it difficult to establish a universally accepted standard for software disclosure. Without a structured approach to balancing these concerns, existing ambiguities may persist, inadvertently enabling selective disclosures and, in extreme cases, academic misconduct. To better understand these challenges, we analyze the priorities and interactions among key stakeholder groups in the research software disclosure landscape.~~

~~We identify four major stakeholder groups in the research software disclosure during the review process: editors, reviewers, authors, and external forces. Their respective priorities and actions, as illustrated in Figure 1A, reveal the potential conflicts shaping this landscape.~~

~~The first major conflict lies between transparency and competitive advantages. Editors and reviewers seek to maximize transparency to enhance the credibility of scientific claims. In contrast, authors, particularly those involved in government-funded, military, or industry collaborations, are often obligated to limit disclosure to protect intellectual property and competitive interests.~~

~~The second tension arises between reproducibility and practical feasibility. While reviewers expect access to fully replicable research environments, meeting these requirements might impose an unreasonable burden on authors, especially when proprietary hardware or software is involved. Moreover, editors must balance the need for transparency with practicality, as journal submission policies cannot impose complete open-source requirements on research in all fields. Notably, different fields have different traditions and research cultures and may be more or less dependent on computational techniques to support their scientific claims (3).~~

~~Lastly, ethical and security concerns can clash with the openness of research software. Some fields face ethical constraints or policy interventions that restrict full disclosure of research software or limit access to select groups. Other concerns, like biosecurity, may pose specific challenges with respect to openness. Curiously, there is an ongoing controversial discussion on the relative merits and disadvantages of sharing the source code of large language models. These concerns have security and geopolitical ramifications. However, excessive restrictions might not only hinder scientific progress but also erode collaboration, ultimately delaying important scientific discoveries.~~


A
preserve competitive advantage in innovation
mitigate risks of technological misuse
external forces
influence the extent of disclosures
maximize the impact and visibility of the research
reduce the burden of disclosure demands
cannot provide to support competitive studies
authors
provide additional disclosure
negotiate the scope of disclosure
multi-stakeholder tensions
evaluate the supportability of research software
balance transparency and feasibility
request additional disclosure
suggest reviewable disclosure
disseminate high-impact research
minimize retraction risks
editors
reviewers
assess the credibility of scientific claims
reduce the burden of peer review process
B
external forces
misguide funding and policy decisions
jeopardize innovation and public trust
create legal and ethical ramifications
misconduct from authors
compromise scientific integrity
damage publisher reputation
waste editorial and peer review efforts
propagate false scientific claims
erode trust in research software
increase difficulty in reproducibility
editors
relevant scholars

~~**Figure 1. Illustration of conflicting priorities among editors, reviewers, authors, and external forces.** (A) The priorities and actions of four stakeholder groups. (B) The impact of misconduct within the stakeholder network.~~

~~Notably, we acknowledge the rationality of these conflicting priorities and our initiative seeks to navigate them, striving to establish an objective and balanced framework for research software disclosure while proactively addressing a critical issue – misconduct – at the research software level.~~

~~In contrast to the conflicting priorities, misconduct introduces a fundamentally asymmetric risk: one in which authors might secure short-term professional gains while editors, relevant scholars (including reviewers), and external forces bear the consequences (Figure 1B). In some cases, authors may exploit these conflicting priorities as a pretext to withhold critical details in research software, rendering their scientific claims unverifiable while still benefiting from the academic prestige associated with academic publication. Such claims, when left unexamined and unchallenged, could mislead external forces, distort funding allocations, and erode public trust in scientific findings. Even with oversight during peer review, editors may lack the technical means to detect these issues, while reviewers, constrained by time, limited access, or, in some cases, a lack of specialized expertise, struggle to verify the presented claims rigorously.~~

~~Overview of Evidence-Oriented Programming~~

~~Compared with the review workflow of general-purpose software (not for publication), there are four unique challenges in research software review:~~

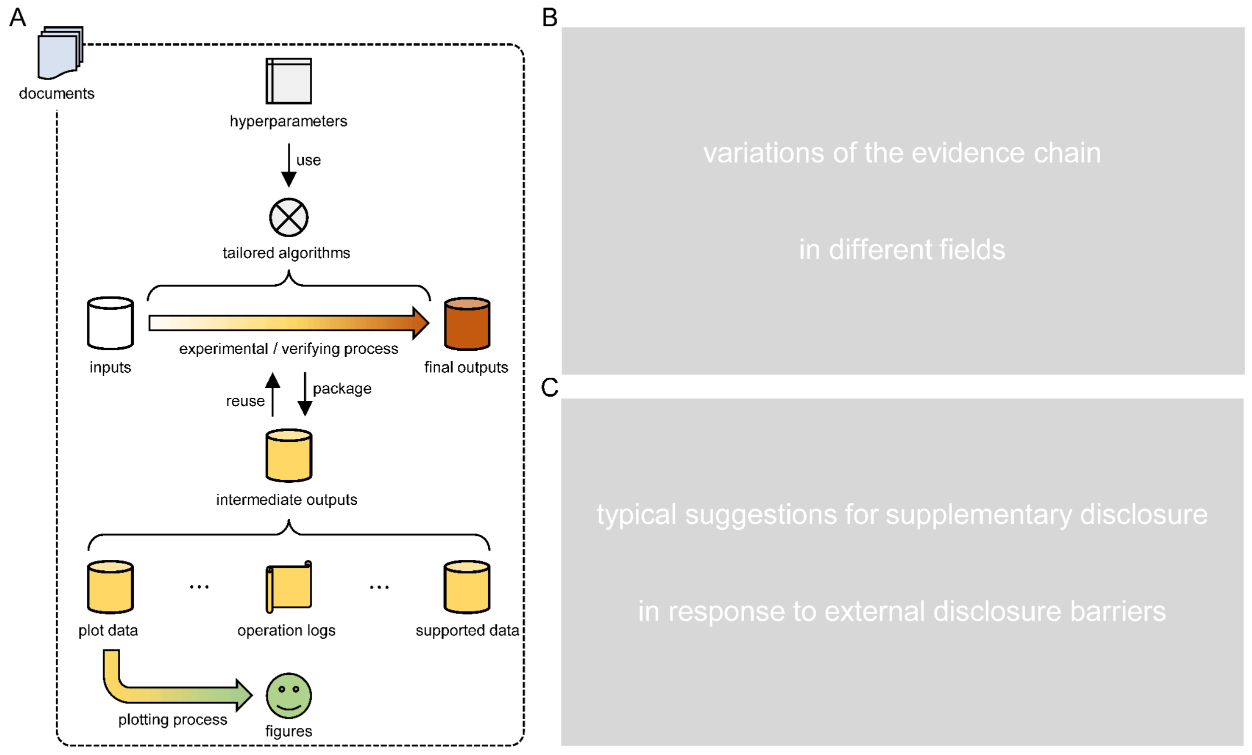
- ~~Research software may incorporate novel data, concepts, and/or uncommon dependencies (whether at the software level or hardware level), potentially bringing greater specificity and heightened complexity to the peer review process;~~
- ~~Due to external disclosure barriers, the source code for partial customized components in research software might require confidentiality, accessible solely through executable files or network endpoints (even limit access times), which hinders the review efficiency;~~
- ~~At times, reviewers may lack the specialized expertise required for a thorough evaluation of research software. When the provided research software is poorly organized, it could further hinder reviewers from providing targeted feedback;~~
- ~~The traditional peer review process, operating under single-blind or double-blind policies, restricts direct interactions between authors and reviewers, thereby complicating efforts to reach consensus on aspects related to research software or causing consensus-reaching considerations to be overlooked.~~

~~Given the unique challenges faced during the peer review process, existing efforts often fall short of ensuring that research software meets evidentiary sufficiency requirements (Supplementary Section 1) – specifically, whether the disclosed components adequately support the scientific claims outlined in the academic publication. Here, Evidence-Oriented Programming introduces structured, indirect interactions between research software developers (i.e., authors) and its evaluators (i.e., reviewers and editors), ensuring that disclosure requirements are negotiated and aligned with both disciplinary norms and external disclosure barriers. Unlike conventional programming paradigms (10), Evidence-Oriented Programming focuses not only on presenting research software but also on establishing a traceable evidentiary link between the software and the scientific claims it supports.~~

~~Specifically, Evidence-Oriented Programming provides a flexible software component disclosure strategy that enables authors, reviewers, and editors to define a minimum disclosure specification for different fields jointly. Furthermore, Evidence-Oriented Programming provides a structured balance for facilitating additional software component disclosures in cases where external disclosure barriers apply. By integrating these principles, Evidence-Oriented Programming enhances the credibility of research software while accommodating the realities of external disclosure barriers in modern scientific research.~~

~~Materialization by Evidence Chain Method~~

~~To put Evidence-Oriented Programming into practice, a structured disclosure method is needed to ensure that accessible components in research software can effectively support scientific claims. Here, we advocate for structuring research software as an evidence chain, as illustrated in Figure 2A. This evidence chain comprises inputs, different processes, intermediate and final outputs (including, but not limited to, process data, result data, plotted data, and operation logs), plotting processes, and figures. Where applicable, tailored algorithms and their hyperparameters can be incorporated into the experimental, verifying, and/or plotting processes. Additionally, each component in the evidence chain should be accompanied by documents or annotations to clarify specific content. By adopting this evidence chain method, authors could ensure a one-to-one correspondence between the presentation of research software and its reported functionality in the corresponding academic publication.~~



~~**Figure 2. Evidence chain method and its variations.** (A) Structure of the basic evidence chain. (B) Variations of the evidence chain in different fields. (C) Typical suggestions for supplementary disclosure in response to external disclosure barriers. A detailed discussion of (B) and (C) can be found in the Supplementary Information.~~

~~The evidence chain can function as a structured linkage between various components in research software, including data and computational processes. By establishing the interdependencies of~~

~~these components, the evidence chain ensures that every element contributing to a scientific claim is accounted for and transparently documented. When a particular component or data element is referenced by a scientific claim but omitted in the disclosure, reviewers or editors are provided with a clear rationale to request its inclusion or seek further clarification from the authors. Furthermore, the evidence chain helps maintain the unidirectional progression from input data to output data, ultimately leading to scientific claims. Reasonable intermediate outputs in the chain can ensure the unidirectionality of progression, reducing the likelihood of generating fake data from scientific claims (11). In essence, this evidence chain method not only strengthens the verifiability of research software but also mitigates the risk of selective disclosures that may undermine the credibility of scientific claims.~~

~~Applicability in Different Academic Fields~~

~~The research software associated with an academic publication does not necessarily need to encompass all components of the evidence chain. The scope of necessary disclosures depends on the software's role within the academic publication and its integration within the broader academic field context (Supplementary Section 2).~~

~~*In progress (next phase).*~~

~~Trade-off Presentation for External Disclosure Barriers~~

~~Here, we suggest introducing additional components to indirectly establish the credibility of undisclosed research software comments when external disclosure barriers prevent their full disclosure (Supplementary Section 3).~~

~~*In progress (next phase).*~~

~~Advancing Adoption through Education and Collaboration~~

~~Some educational contents are shown in Supplementary Section 4~~

~~*In progress (next phase).*~~

~~Conclusions~~

~~The disclosure of research software should be designed to validate scientific claims efficiently, prioritizing its role as a research artifact over its technical characteristics as software. Here, Evidence-Oriented Programming provides the conceptual foundation for this shift. Furthermore, the evidence chain method offers a structured development to align software components with corresponding scientific claims. It enhances transparency and reproducibility and accommodates necessary disclosure constraints, striking a rational balance between openness and practical feasibility. To advance its adoption, further refinement and multi-party discussions are needed to integrate the concept of evidence-oriented programming into research software developments and peer review processes. Establishing clear expectations and incentivizing structured software disclosures will be essential to fostering a research ecosystem where research software is not just a computational tool but a verifiable supplementary of scientific inquiry.~~

~~Acknowledgments~~

~~Waiting until the final moment for an update.~~

~~References and Notes~~

1. ~~Morane Gruenpeter, *et al.*, *Zenodo*, 5504016 (2021).~~
2. ~~Michelle Barker, *et al.*, *Sci. Data* **9**, 622 (2022).~~
3. ~~Haoling Zhang, *et al.*, *Comput. Struct. Biotechnol. J.* **23**, 3989–3998 (2024).~~
4. ~~Gregory D. Poore, *et al.*, *Nature* **579**, 567–574 (2020).~~
5. ~~Nathan Dasenbrock-Gammon, *et al.*, *Nature* **615**, 244–250 (2023).~~
6. ~~Robert Heilmayr, *et al.*, *Science* **382**, 1171–1177 (2023).~~
7. ~~G. Stefano Brigidi, *et al.*, *Cell* **179**, 373–391 (2019).~~
8. ~~Mandeep R. Mehra, *et al.*, *N. Engl. J. Med.* **382**, e102 (2020).~~
9. ~~Editorials, *Nature* **629**, 728 (2024).~~
10. ~~Devon M. Simmonds, *Computer* **45**, 93–95 (2012).~~
11. ~~Miryam Naddaf, *Nature* **623**, 895–896 (2023).~~

Supplementary Information

Systematic Comparison with Early Efforts

Framework-level Comparison with Evidence-Oriented Programming

To clarify the complementary role of Evidence-Oriented Programming (EOP) in the broader landscape of research software governance, here we compared EOP with several well-established frameworks (1-5), including FAIR4RS, the Software Sustainability Institute (SSI) Guidance, Collective Benefit, Authority to Control, Responsibility, Ethics (CARE) principles for indigenous data governance, the Transparency and Openness Promotion (TOP) guidelines, and Software Engineering for Science (SE4Science). Each framework has made critical contributions to improving software management, discoverability, sustainability, and ethical governance, laying a strong foundation for responsible software practices in science.

FAIR4RS stands out for its comprehensive focus on ensuring that research software is findable, accessible, interoperable, and reusable, greatly enhancing its long-term visibility and potential for reuse. However, FAIR4RS does not explicitly focus on how software disclosures should support scientific claims during peer review – a critical gap EOP aims to fill. While FAIR4RS optimizes the software's future utility, EOP ensures that software disclosures directly contribute to claim verification at the time of publication.

The SSI guidance is equally important in promoting best practices for software sustainability and ensuring that research software remains maintainable, extensible, and valuable over the long term. Its primary audience, developers and project leads, benefits from practical advice on project governance and technical management. Yet, the SSI guidance does not explicitly address how research software should be disclosed in the peer review process. EOP complements the SSI guidance by introducing a structured evidentiary approach that enables reviewers and editors to assess the credibility of scientific claims through research software.

The CARE principles have made vital advances in embedding ethical and community-centered governance into data practices, particularly in protecting the rights and sovereignty of Indigenous communities. While these principles address data governance comprehensively, they do not target the role of research software in scientific peer review. EOP, while fully compatible with ethical governance values, focuses specifically on enhancing software credibility within the academic publishing process.

The TOP guidelines have driven a significant cultural shift toward transparency across the research lifecycle, encouraging clearer reporting of methods, data, and materials. However, the guidelines primarily address broad transparency rather than the evidentiary role of research software itself. EOP extends this agenda by introducing the evidence chain structure, ensuring that software disclosures are explicitly linked to the scientific claims they support, and making computational steps traceable and verifiable during peer review.

SE4Science has advanced software engineering best practices tailored to scientific software, including version control, testing, modularity, and continuous integration recommendations. These practices are essential for ensuring robust, reliable, and maintainable software. However, SE4Science does not explicitly address the evidentiary role of software in peer review, nor does it provide a structured process for negotiating disclosure between authors, reviewers, and editors. EOP directly fills this procedural and evidentiary gap.

These existing frameworks offer critical building blocks for improving research software management, ethical governance, and technical sustainability, but none directly treats research software as structured scientific evidence essential to the credibility of scientific claims during peer review. EOP builds upon and complements these earlier efforts by addressing the missing linkage between software component disclosure and scientific claim verification, providing a review-embedded and field-adaptive framework designed to enhance transparency, reproducibility, and accountability in research software disclosure.

Technology-level Comparison with Evidence Chain Method

A large number of existing technologies – exemplified by Sumatra, Taverna, myExperiment, Common Workflow Language, Docker, noWorkflow, ReproZip, Jupyter Notebooks, Binder, Whole Tale, Code Ocean, and Renku – have significantly advanced the transparency and reproducibility of computational research (6-17). They embody a technological family that emphasizes capturing, packaging, sharing, and re-executing computational workflows, ensuring reviewers and future researchers can technically replicate prior analyses. Their contribution to technical reproducibility forms an essential pillar of responsible computational research.

These existing technologies shared several core methodological features. Most focus on capturing entire computational environments, including code, dependencies, configurations, and data. By encapsulating the runtime context, they reduce the risk of configuration drift, ensuring that analyses can be re-executed even as software ecosystems evolve. Additionally, several tools emphasize workflow documentation and sharing, allowing researchers to define analytical steps and promote collaborative reuse. However, despite their strengths in preserving computational artifacts, these technologies do not explicitly address the evidentiary role of research software in supporting scientific claims during peer review. They focus primarily on the ability to re-run a computational process, rather than on structuring software disclosures to demonstrate their role in substantiating each scientific claim made in a publication. Furthermore, they offer limited mechanisms for reviewers and editors to request additional disclosures or negotiate customized requirements based on the specifics of the study and its disciplinary context.

The evidence chain method complements existing technologies by introducing a claim-centered disclosure structure, in which each disclosed software component is explicitly linked to the scientific claim it supports. This method is not just a tool for preserving processes; it functions as a structured evidentiary map, ensuring that research software serves as traceable scientific evidence. By embedding this structure directly into the peer review process, this method also facilitates author-reviewer-editor negotiation, allowing reviewers to request disclosures that are proportional to the evidentiary role of research software in the study. More importantly, this method introduces field-adaptive flexibility, recognizing that disclosure expectations vary across disciplines. This contrasts with many existing technologies, which apply uniform technical standards regardless of disciplinary context. This method can further accommodate external disclosure barriers, by allowing for substitutional evidence.

While existing technologies have been instrumental in advancing reproducibility, they leave a critical gap in scientific verifiability and claim-centered transparency during the peer review process. The evidence chain method fills this gap, not by replacing these tools, but by introducing a complementary layer of evidentiary structure, designed to align research software with the scientific claims it supports and to ensure that disclosure practices are both negotiable and adaptable across different fields and regulatory contexts.

Detailed Case Usage in Different Fields

We examine several academic fields as case studies to discuss the typical disclosure requirements for components in research software. Given the increasing overlap between academic fields, research software development and presentation in interdisciplinary studies should be guided by the primary field it most closely aligns with.

High Performance Computing (mainly contributed by Prof. David Keyes)

In progress (next phase).

Machine Learning (mainly contributed by Prof. Philip Torr)

In progress (next phase).

Cryptography (mainly contributed by Prof. Yu Yu)

In progress (next phase).

Computational Material Science (mainly contributed by Dr. Julien Gorenflot)

In progress (next phase).

Computational Chemistry (mainly contributed by Chen Lin)

In progress (next phase).

Bioinformatics (mainly contributed by Prof. David Gomez-Cabrero)

In progress (next phase).

Synthetic Biology (mainly contributed by Dr. Yue Shen)

In progress (next phase).

Detailed Case Usage in Different External Disclosure Barriers

XX

Commercial Pressure

We need to identify what the core components in research software of commercial protection are.

In progress (next phase).

Technical Misuse Concern

We need to identify which components of research software, if disclosed, could lead to technical misuse.

In progress (next phase).

Data Sensitivity Issue

We need to define which software components are prone to sensitive data being captured.

In progress (next phase).

Periodic Suggestions for Beginners

For beginners, especially researchers without formal training in research software development, directly creating research software with a fully embedded evidence chain can be particularly challenging. To address this, we provide a set of suggestions that accompany the development process itself, offering practical guidance at different development stages. These suggestions are designed to gradually introduce key practices, helping beginners to create research software that incorporates evidence attributes. This stepwise approach not only lowers the initial learning barrier but also cultivates essential awareness of how to make research software reviewable, reproducible, and accountable.

Initializing a Structured Archive

The most fundamental awareness of evidence chaining begins with how to reasonably structure the research software archive. A well-organized archive not only enhances the clarity and maintainability of the code repository but also serves as the foundation for linking scientific claims to corresponding files in the archive.

It is encouraged to adopt clear directory conventions as Figure S1, which provides a template suitable for most research projects. Depending on the actual needs of a given research project, this structure can be further adapted, e.g., by adding folders for execution logs or configuration files. The key principle is to ensure that every critical element contributing to the reproducibility and evidence traceability of the research is explicitly captured within the archive, making it self-contained and understandable to others, including future collaborators and reviewers.

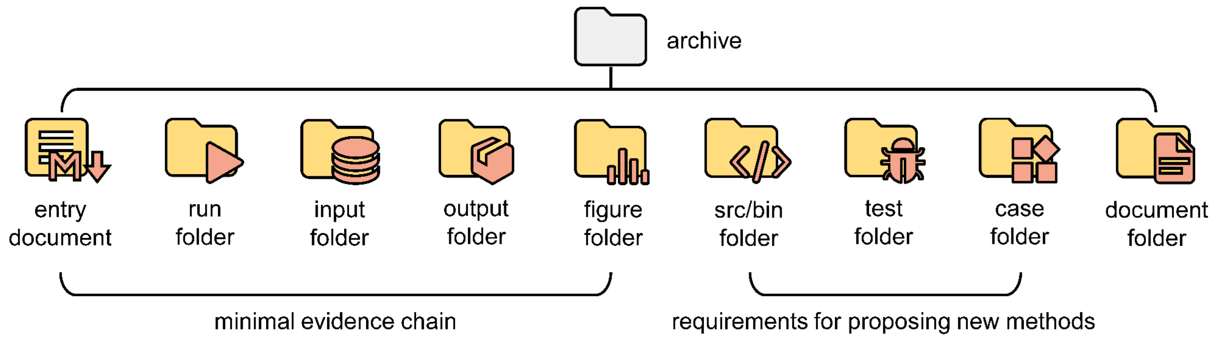


Figure S1. Suggested file structure in the archive. It is reported from (18).

In practice, some scripts responsible for experimental, verifying, and/or plotting processes may require iterative updates as the research project evolves, such as adding new processing steps or adjusting analysis logic. In such cases, we recommend recording these iterations either by embedding timestamps into file names like "analysis_2025.march.4.py" or by relying on a version control system like GitHub to manage the entire revision history. Although these recorded scripts might not necessarily be shared with reviewers or editors during the peer review process, they play a crucial role in maintaining research software as the research project evolves.

Reorganizing Experimental, Verifying, and/or Plotting Process Scripts

The scripts responsible for experimental, verifying, and/or plotting processes occupy a position in the archive equally important as tailored algorithms (if applicable, or otherwise these scripts become the primary focus), because they serve as the backbone of the evidence chain -- the critical link connecting raw data, analysis, and reported results.

To ensure full transparency and traceability, these processes must be presented in a manner that maps directly and explicitly to every data handling step described in the manuscript. Such direct mapping ensures that each transformation, filtering, and processing action is documented not only in narrative form but also in executable form within the code.

Consequently, once the computational process within a script is finalized, it becomes essential to reorganize or refine the script to enhance its correspondence with the manuscript's procedural descriptions. Annotations and structural markers can be made to segment the code according to the manuscript's methodological steps. This refinement not only enhances code clarity but, more importantly, reinforces the script's function as a verifiable procedural mirror to the reported methods and results.

Preparing Entry Document to Associate Different Folders

The entry document in the archive serves a role essentially equivalent to that of an abstract in a manuscript. In essence, it provides a concise summary of the roles different files from the archive play in supporting the scientific claims outlined in the manuscript. It covers, though is not confined to, the following aspects: an overview of the software's intended purpose and key functionalities, guidelines for configuration and deployment, an inventory listing all files within the archive, and guidance on how to run or execute the research software.

The process of drafting this entry document typically signals that the research project is nearing completion, with materials being prepared for submission to a journal or conference. The first essential step in this process is to thoroughly review all files in the archive, identifying which files should be included in the final submitted version, and which files are redundant or unrelated byproducts of research processes. Building on this initial organizing, beginners are also expected to trace and document the specific relationships between files – clarifying where each file originated, how it was produced and how it was used if applicable. For larger files or those with weaker relevance to the research outcomes, it may be more appropriate to store them in an external location. In such cases, a checksum or verification record (e.g., MD5 value of the file) should be included in the entry document to facilitate inspection by reviewers if needed.

References and Notes

1. Michelle Barker, *et al.*, *Sci. Data* **9**, 622 (2022).
2. Stephen Crouch, *et al.*, *Comput. Sci. Eng.* **15**, 74–80 (2013).
3. Stephanie Russo Carroll, *et al.*, *Data Sci. J.* **19**, 1–12 (2020).
4. Brian A. Nosek, *et al.*, *Science* **348**,1422–1425 (2015).
5. Jeffrey C. Carver, Neil P. Chue Hong, and George K. Thiruvathukal, *CRC Press* (2016).
6. Andrew Davison, *Comput. Sci. Eng.* **14**, 48–56 (2012).
7. Tom Oinn, *et al.*, *Bioinformatics* **20**, 3045–3054 (2004).
8. Carole A. Goble, et al., *Nucleic Acids Res.* **38**, W677–W682 (2010).
9. Michael R. Crusoe, et al., *Commun. ACM* **65**, 54–63 (2022).
10. Dirk Merkel, *Linux J.* **2014**, 2 (2014).
11. Leonardo Murta, et al., In *Proc. 5th Int. Provenance and Annotation Workshop*, 71-83 (2014).
12. Fernando Chirigati, *et al.*, In *Proc. 2016 Int. Conf. Manage. Data*, 2085–2088 (2016).
13. Thomas Kluyver, *et al.*, In *Proc. 20th Int. Conf. Electron. Publ.*, 87–90 (2016).
14. Matthias Bussonnier, *et al.*, In *Proc. 17th Python Sci. Conf.*, 113–120 (2018).
15. Adam Brinckman, *et al.*, *Future Gener. Comput. Syst.* **94**, 854–867 (2019).
16. Thomas Staubitz, *et al.*, In *2016 IEEE Glob. Eng. Educ. Conf.*, 314–323 (2016).
17. Rok Roškar, et al., In *Adv. Neural Inf. Process. Syst. 36*, 1–13 (2023).
18. Haoling Zhang, *et al.*, *Comput. Struct. Biotechnol. J.* **23**, 3989–3998 (2024).