Research software as scientific evidence

Developing research software is a technical task – how it is appropriately disclosed is a scientific matter.

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 transparency, accessibility, and reusability in research software by establishing guidelines, recommending good practices, developing verification platforms, and offering educational resources (2–6). 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 already had real-world consequences. One notable example is *Nature*'s decision to publish the AlphaFold3 paper without its code (7). Specifically, *Nature Portfolio Editorial Policies* require that code supporting central claims 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. *Nature* later cited biosecurity risks and the inclusion of pseudocode to justify the editorial decision, while *DeepMind* attributed the restriction to commercial considerations. This example illustrates how each stakeholder operates from a distinct position and how the absence of shared expectations can lead to tensions in disclosure decisions. Informed by extensive consultations across the stakeholder community, a detailed illustration of stakeholder interactions within the research software disclosure ecosystem is shown in Fig. 1.

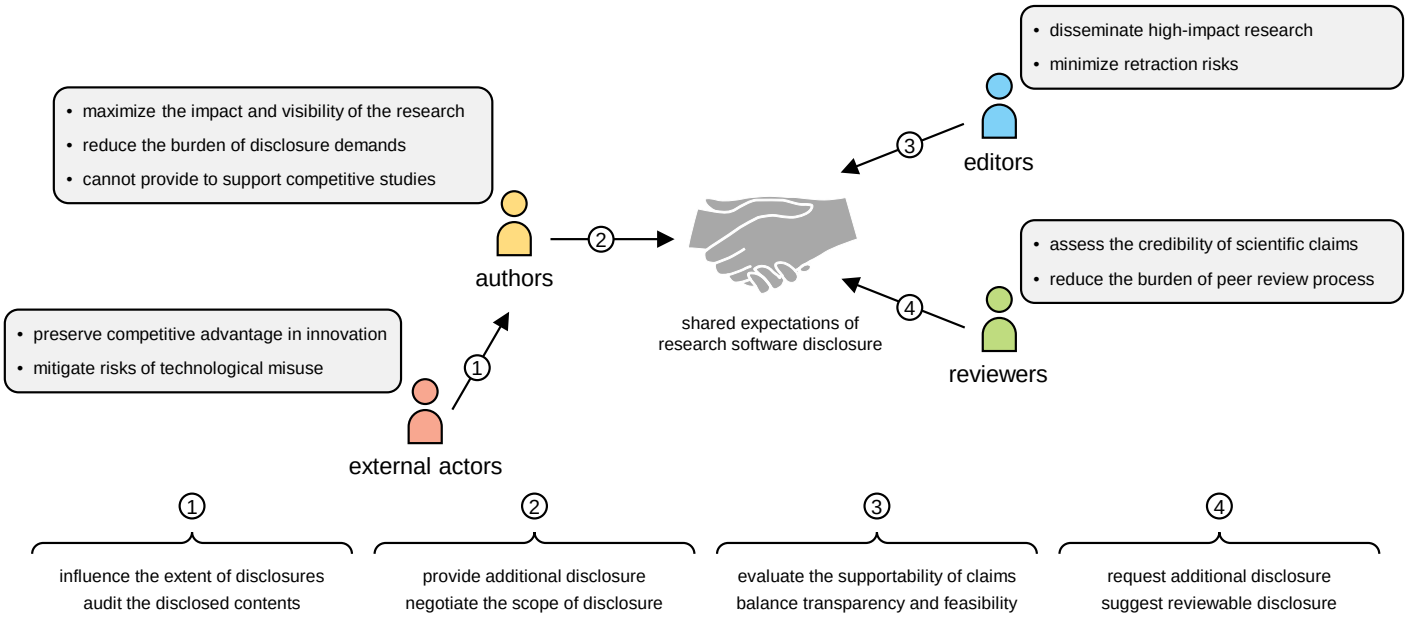

Fig. 1 | **Stakeholder interactions within the research software disclosure ecosystem.** Rounded boxes indicate positions, and arrows represent behaviors. External actors include, but are not limited to, regulatory agencies, funding bodies, and other entities that provide oversight or guidance on disclosure practices.

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 (8). 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 security concerns, all of which introduce competing priorities that shape what can be disclosed. Together, these factors constrain the broader establishment of shared disclosure specifications, making it increasingly difficult to keep pace with the growing complexity of contemporary research practice.

Herein, we report the outcome of discussions involving representatives from the stakeholder community that helped initiate the preliminary development of shared disclosure specifications. Building on insights synthesized in prior reviews of the evolving history of software governance (8), together with clarifications of responsibilities and incentives across different stakeholder groups, we propose a new conceptual model, termed *Evidence-Oriented Programming*, to inform future governance. Rather than focusing on how research software is technically implemented, this model reorients software

development and evaluation around the relationship between scientific claims and computational artifacts (i.e., software components and data elements) that substantiate them. To translate this conceptual shift into practice, we further introduce the *Evidence Chain Formalization* as an operational strategy for identifying, structuring, and linking the computational artifacts necessary to support the scientific claims in a publication. We discuss its applicability across different fields and propose how it may help mitigate 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 TOP Guidelines (2), the FAIR4RS Principles (3), and the COAR Framework (9). Therefore, 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 (10). A successful replication may simply reproduce unrecognized systematic errors (11). 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% (12).

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 (11).

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.

- **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. 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. 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.
- **Timing:** In most cases, authors can make research software fully accessible at the time of submission, and we encourage them to do so. Yet, 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 (13). 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.
- **Form:** Evaluating research software poses challenges distinct from general-purpose software review (14). 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 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.

Evidence Chain Formalization

The Evidence Chain Formalization (ECF), which we developed by adapting ideas from scientific workflows (15), 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. ECF responds to this challenge by mapping computational artifacts to the claim they support, making evidentiary structures explicit, interpretable, and available for review.

ECF 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 information. The ending artifacts are the resulting *visual claims*: figures, tables, and statistical quantities such as correlation coefficients or confidence levels that readers interpret as evidence in the publication. This range makes clear that ECF 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 ECF.

ECF provides a basic evidence chain to link the starting and ending artifacts (Fig. 2). First, *input data* are transformed into output data through the *experimental or analytical process(es)*. The *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. An illustrative example of EOP/ECF-compliant research software, drawn from a recent peer-reviewed academic publication, is provided in the supplementary materials.

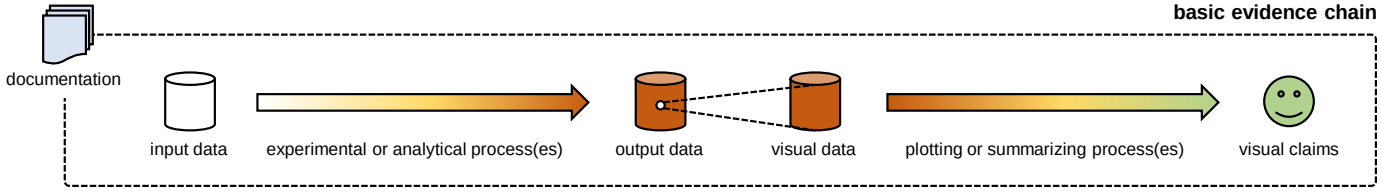

Fig. 2 | **Structure of the basic evidence chain.** This artifact structure formalization enables evaluation of evidentiary sufficiency within the software-mediated procedural range. Two process-oriented computational artifacts vary with research contexts and external disclosure barriers (detailed in supplementary materials).

While the above seven artifacts form a basic evidence chain, different academic fields or differences in the strength or type of scientific claims can introduce additional complexity, especially within the *experimental or analytical process(es)*. These differences may range from simple software invocations to sophisticated multi-stage software call pipelines. Further, where applicable, additional data elements such as hyperparameter configurations and random seeds may also become critical for evidentiary interpretation. ECF can accommodate such variation (as detailed in the supplementary materials), ensuring that scientific claims remain evidentially sound even amid research-context heterogeneity and claim-level differences.

In real-world settings, full disclosure of evidentiary computational artifacts may not be feasible due to ethical, legal, security, or intellectual property constraints. ECF supports the decoupling between evidentiary verifiability and artifact accessibility to a limited extent, and enables supplementary disclosure strategies that preserve evidentiary traceability without requiring unrestricted access to all underlying computational artifacts. These mechanisms (detailed in supplementary materials) help reviewers and editors assess scientific claims when direct disclosure is constrained.

Overall, the above software disclosure suggestions in ECF collectively instantiate the *scope* and *form* dimensions of EOP (detailed in supplementary materials). To extend ECF 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.

Practical Challenges and Corresponding Solutions

Although ECF provides a concrete operationalization of EOP, its practical adoption faces challenges on several fronts. On the implementation side, ECF places substantial demands on authors: it requires proficiency in software design, documentation, and the ability to articulate software components in evidentiary form – skills that many researchers without formal software-engineering backgrounds may lack. On the evaluation side, ECF remains in an early stage: field-specific variants have not yet been systematically developed, and no large-scale studies currently assess its evidentiary effectiveness across academic fields. At the infrastructural level, current publishing workflows and quality-assurance practices are not designed to support evidence-chain review, and the storage capacity typically provisioned for scientific projects is far below what is required to host the intermediate artifacts and evidentiary traces demanded by EOP/ECF-compliant software implementations.

To address the implementation challenges, we are constructing development guidelines that reflect how researchers actually engage with software in their respective academic fields. In addition to static tutorials, we are continuously engaging in one-to-one discussions and workshop-style exchanges with researchers from different academic fields and with varying levels of software development experience, in order to support clearer understanding and practical adoption among the intended audiences. Both strands of these efforts are reflected, to some extent, in the supplementary materials. We are also testing ways to embed ECF into existing scientific activities, including exploring whether AI agents might assist researchers in identifying and annotating evidentiary artifacts during software development (in collaboration with *NVIDIA Research*).

To support evaluation practices, we are working with software-quality-assurance communities and academic publishers to build the structural conditions necessary for ECF to take root. This includes developing editorial guidelines for reviewers and editors, and partnering with publishing platforms to prototype features that surface evidence-chain structure. These efforts aim not only to make ECF technically feasible but also to integrate evidentiary review into publishing workflows in ways that are practical and scalable.

To strengthen the infrastructural layer, we are addressing the data-management demands introduced by EOP/ECF-compliant implementations. As illustrated in our supplementary example, such practices generate substantial intermediate artifacts and evidentiary traces. These can be hosted on established research data repositories such as *Figshare* or institutional equivalents, ensuring their long-term accessibility and reproducibility.

Potential exceptional situations

Beyond these ongoing efforts, certain exceptional situations may require measures that fall outside routine EOP practice. We suggest that, prior to manuscript acceptance, authors predefine which designated audiences (e.g., regulatory bodies, ethics boards, or expert oversight panels) should be granted full access to their research software under clearly specified conditions – namely, when the work attracts widespread attention and becomes the subject of substantial technical doubt after publication. Although this proposal extends beyond the technical scope of EOP itself, it addresses adjacent concerns about evidentiary accountability and might merit 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 ECF 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 evaluation.

We hope this work will serve as a catalyst for broader engagement across the stakeholder community in developing shared specifications around evidentiary sufficiency. Supported by incentives for structured software disclosures, such specifications 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. As a first step, we encourage publishers, funders, and oversight bodies to pilot evidentiary specifications for research software disclosure within existing review and assessment workflows, enabling iterative refinement grounded in real editorial and evaluative practice.

References and Notes

1. M. Gruenpeter et al. *Defining research software: a controversial discussion. Zenodo* (2021).
2. B. A. Nosek et al. Promoting an open research culture. *Science* **348** (2015), 1422–1425.
3. M. Barker et al. Introducing the FAIR principles for research software. *Scientific Data* **9** (2022), 622.
4. H. Hunter-Zinck et al. Ten simple rules on writing clean and reliable open-source scientific software. *PLoS Computational Biology* **17** (2021), e1009481.
5. T. Staubitz et al. CodeOcean – A versatile platform for practical programming excercises in online environments. *IEEE Global Engineering Education Conference*. (2016), 314–323.
6. G. Wilson. Software carpentry: getting scientists to write better code by making them more productive. *Computing in Science & Engineering* **8** (2006), 66–69.
7. S. Wankowicz et al. *AlphaFold3 Transparency and Reproducibility. Zenodo* (2024).
8. H. Zhang et al. Reviewability and Supportability: New complementary principles to empower research software practices. *Computational and Structural Biotechnology Journal* **23** (2024), 3989–3998.
9. Confederation of Open Access Repositories. *COAR Community Framework for Good Practices in Repositories, Version 2. Zenodo* (2022).
10. National Academies of Sciences and Medicine, Policy and Global Affairs, Board on Research Data and Information, et al. *Reproducibility and replicability in science*. National Academies Press, 2019.
11. G. Miller. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science* **314** (2006), 1856–1857.
12. A. Trisovic, M. K. Lau, T. Pasquier, and M. Crosas. A large-scale study on research code quality and execution. *Scientific Data* **9** (2022), 60.
13. J. M. DuBois. *RCR Casebook: Stories about Researchers Worth Discussing*. U.S. Office of Research Integrity. (Visited on 06/16/2025).
14. N. B. Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* **35** (2010), 8–13.
15. C. S. Liew et al. Scientific workflows: moving across paradigms. *ACM Computing Surveys* **49** (2016), 1–39.

Supplementary Materials for

Research software as scientific evidence: clarifying missing specifications

Haoling Zhang *et al.*

Corresponding authors: Haoling Zhang (haoling.zhang@kaust.edu.sa) and Jesper N. Tegnér (jesper.tegner@kaust.edu.sa)

This PDF file includes:

- Author Affiliations, Contributions, and Acknowledgments
- Clarifications on Frequently Raised Questions about this Initiative
- A Peer-reviewed EOP/ECF-compliant Software Case Study
- Stage-wise Software Development Suggestions
- Evidence Chain Differentiation across Research Contexts
- Evidentiary Verifiability Decoupling and Supplementary Disclosure under External Disclosure Barriers
- One-to-one Interviews of this Initiative

Abbreviations are:

- EOP: Evidence-Oriented Programming
- ECF: Evidence Chain Formalization

Author Affiliations, Contributions, and Acknowledgments

Clarifications on Frequently Raised Questions about this Initiative

Here, we summarize several topics that have been repeatedly raised in our internal discussions and provide corresponding clarifications.

Why is EOP necessary beyond the existing software initiatives?

EOP differs from existing software-related efforts in that it begins with a more explicit analysis of the nature and function of research software within the academic context. Rather than treating research software solely as an engineering artifact, we examine its role as a generator of scientific evidence. This motivates the introduction of evidentiary sufficiency as a complementary concept to replicability: while replicability captures whether results can be reproduced under identical conditions, evidentiary sufficiency concerns whether the disclosed processes provide enough information for readers to evaluate the evidentiary support behind scientific claims. In our view, this fills a critical gap in current research software discussions.

A central premise of EOP is that full disclosure of research software is often constrained. These constraints are not exceptional but structural and widespread, arising from heterogeneous priorities across stakeholder groups. The primary purpose of EOP is to acknowledge the legitimate and often divergent needs of different stakeholders regarding software disclosure, and to navigate these interests toward a workable and minimally sufficient level of consensus. By doing so, EOP provides a shared disclosure context within which the reproducibility properties of research software can be meaningfully leveraged in assessing the reasonableness of a manuscript's scientific claims, or, at the very least, in improving the evidentiary sufficiency with which those claims are supported.

Accordingly, the contribution of EOP lies not in prescribing universal forms of software openness or in standardizing development practices, but in providing a conceptual model for coordination and a set of operational principles for shaping evidentiary transparency. The goal is to identify which components of research software are essential for evaluating scientific claims, how these components can be disclosed under realistic constraints, and how the resulting evidence can be organized.

What does ECF contribute within EOP?

To operationalize the coordination aims of EOP, we distinguish three dimensions of research-software disclosure: scope, timing, and form. Because these dimensions are shaped not only by heterogeneous priorities across stakeholder groups but also by research contexts and external disclosure barriers, no universally applicable disclosure standard is feasible.

To bypass the impasse created by such diversity, we instead approach the problem from the perspective of research software as evidentiary support for a scientific manuscript. In a scientific manuscript, the central scientific claims are conveyed through visual claims (i.e., figures, tables, and statistical quantities). However, the reliability of a visual claim cannot be established merely by confirming the existence of the individual computational artifacts (i.e., data elements or software components) used in producing it. Each visual claim is the output of a sequence of data transformations that may include ingestion, preprocessing, modeling, filtering, statistical aggregation, and visualization. Although these artifacts are technically connected within a workflow, their evidentiary relationships are rarely visible.

ECF directly addresses this problem by reconstructing the evidence chain(s) from raw or generated input data to the resulting visual claims. This reconstruction reveals how each computational step contributes to the reported result, allowing evaluators to determine whether the operations are appropriate for the scientific question and whether the visual claim is adequately supported. Through this structure, ECF converts visual claims from opaque endpoints into auditable evidentiary objects.

Building on this evidence-chain structure, ECF can provide practical guidance for the three dimensions of EOP, i.e., scope, timing, and form, introduced above. First, by specifying the transformation structure that links inputs to visual claims, ECF clarifies the *form* in which research software should be disclosed: not as an undifferentiated code repository, but as a set of computational artifacts that are explicitly connected through their evidentiary roles. Second, this structure enables principled judgments about disclosure *scope*. Because individual artifacts can be selectively removed within the chain, evaluators can determine whether the omission of a particular component would alter the resulting visual claim or undermine the central scientific claims. Artifacts whose removal does not affect evidentiary adequacy may be safely omitted from disclosure, whereas those whose absence compromises the evidence chain need to be considered for retention. Finally, this structure also informs decisions about disclosure *timing*. In situations where premature release of certain artifacts could lead to misuse (like competitive disadvantage or inappropriate replication), this structure permits these artifacts to be disclosed at later stages, provided that the remaining structure is sufficient for partial evidentiary evaluation during peer review. Overall, these functions allow ECF to support flexible yet principled disclosure practices that align with the coordination aims of EOP.

Can EOP and ECF be used across all scientific activities?

The applicability of EOP and ECF depends on whether the scientific manuscript relies on research software to generate its visual claims or other evidentiary outputs. When computational artifacts play no role in the production of results, e.g., in manuscripts based solely on photographic evidence, manual experimental procedures, or purely theoretical and mathematical derivations, the evidentiary structure does not contain the kind of software-mediated transformations that EOP and ECF are designed to analyze.

Why is ECF beneficial to the community's diverse stakeholder groups?

By creating a structured evidentiary basis for discussion, ECF brings otherwise competing or seemingly incompatible requests into a negotiable space, allowing them to be addressed without imposing uniform requirements on all parties. As a result, ECF has the potential to create more shareable and predictable software-disclosure expectations for the community as a whole, enabling each stakeholder group to benefit in different but complementary ways. Below, we outline the specific advantages for each stakeholder group:

- **Authors:** (1) ECF can strengthen authors' ability to self-audit the processes involved in generating figures, tables, and statistical quantities. During the iterative and exploratory stages of a research project, code versions of research software often evolve as the definition of the research object changes, as analytical decisions shift, and as the targets of the visual claims are updated, making errors, inconsistencies, or undocumented modifications more likely to arise. By improving the traceability of these evolving computational artifacts, ECF helps authors detect such issues earlier. (2) ECF can reduce the risk of misunderstandings during peer review by providing authors with a defensible rationale for the scope and timing of their software disclosure. This is particularly valuable in research projects that face external disclosure barriers, where the initiative helps authors more effectively balance evidentiary sufficiency with the practical limits of transparency.

- **Reviewers:** (1) ECF can help prevent reviewers from being sidetracked by low-relevance or peripheral software implementation details, because it provides a more structured evidentiary architecture for research software. For instance, in some research projects, research software may

accumulate several versions of a script that implement similar or partially overlapping functionality. Without a structured evidentiary architecture, reviewers may find it difficult to determine which version contributed to the construction of the visual claims, increasing the likelihood of confusion or misinterpretation. (2) ECF can improve the evaluability of computational tasks by making the processes that generate visual claims more traceable. Under current practice, reviewers often have to infer how the authors carried out key steps, because the relevant analytical and computational processes are not always explicitly visible. By enabling reviewers to follow how inputs are transformed into visual claims, they can judge more directly whether the evidence presented is sufficient to support the scientific claims made in the manuscript. As a result, this reduces reliance on guesswork and increases both the efficiency and the reliability of peer review.

- **Editors:** ECF can offer editors a clearer and more quantifiable basis for assessing the implications of missing or incomplete software-related information. When certain computational artifacts are absent, under-specified, or difficult to contextualize, it can be challenging for editors to judge whether such omissions materially affect the evidentiary strength of the manuscript's scientific claims. In combination with reviewers' reports, especially when these include comments on software-related aspects of the manuscript, this evidentiary clarity helps editors communicate more precisely with authors about what additional disclosure or clarification may be needed. Hence, ECF could help editors anchor their guidance in specific evidentiary needs, thereby improving both the efficiency and the precision of editor-author communication.

- **External actors:** ECF can benefit external actors whose expectations about transparency may not be identical to those of the broader academic community. By making the evidentiary core of software disclosure explicit, ECF partitions disclosure expectations into an evaluation layer defined by, and extendable from, the basic evidence chain and geared toward scholarly assessment, and an oversight layer that may be motivated by compliance, risk management, or public-interest considerations. This layered framing helps reduce direct competition over a single, undifferentiated notion of transparency: external actors can treat the evaluation layer as necessary and focus their requests on clearly articulated, purpose-specific adjustments when justified.

It is important to note, however, that these benefits do not imply the absence of practical challenges. As clarified in the main text, our expert consultations and preliminary tests highlight several challenges, along with ongoing or prospective strategies for addressing them. Therefore, while ECF can improve evidentiary sufficiency and help navigate differing stakeholder priorities, its adoption will necessarily proceed through iterative refinement grounded in community feedback.

How does ECF differ from scientific workflows?

Both ECF and scientific workflows (1) operate on chain-like structures and both apply to research software. However, despite this superficial similarity, they differ fundamentally in purpose. ECF is a conceptual approach for tracing back from a scientific claim to the minimal set of artifacts required to support it, and for assessing whether those components have been adequately disclosed and contextualized. In contrast, scientific workflows are operational approaches that describe the concrete sequence of steps or data transformations used to execute an analysis or computation. They are excellent for documenting execution, but they do not determine whether each artifact is evidentially relevant or sufficient to support a specific claim. Although conceptually distinct, ECF and scientific workflows are mutually reinforcing: scientific workflows can benefit from ECF by gaining a clearer evidentiary target, while ECF can benefit from scientific workflows through the automated execution and inspection of evidence chains.

Does ECF address scientific misconduct?

ECF is not designed to detect or prevent scientific misconduct. While some recent retraction cases have shown that certain forms of misconduct eventually come to light, such detections typically originate from the domain expertise and professional judgment of readers (2, 3), rather than from software-disclosure practices. The visibility provided by ECF can, in some cases, make such detections easier. Nevertheless, ECF alone may not be sufficient to detect intentional wrongdoing during the peer review stage.

Moreover, if misconduct arises from a deliberate intent to deceive and the authors possess even moderate software-development skills, ECF can in principle be circumvented. One could route computations through external data sources, external software components, or other nondeterministic systems outside the disclosed environment. Such behaviors fall outside the control of any disclosure framework and cannot be reliably prevented by technical means alone.

What ECF can achieve is to strengthen the conditions for both self-evaluation by authors and external evaluation by reviewers and editors. By ensuring that the transformations leading to visual claims are traceable and that key computational artifacts are auditable, ECF reduces evidentiary ambiguity and helps expose inconsistencies that arise from incomplete or poorly structured evidence. In this sense, ECF does not eliminate misconduct, but it narrows the space in which evidentiary insufficiency can remain unnoticed.

Does ECF ensure the irreversibility of visual-claim generation?

In standard scientific practice, computational tasks proceed in a forward direction: from input data, through a sequence of transformations, to visual claims. This directionality reflects the evidentiary logic of scientific reasoning, where visual claims are summaries of well-defined upstream processes. However, recent discussions have revealed the possibility of reverse-engineered (or output-driven) generation (4, 5), in which the final result influences, or even partially determines, the construction of upstream inputs. These developments challenge the assumption that visual claims always originate from a strictly forward, input-anchored workflow.

ECF does not by itself ensure that visual-claim generation is irreversible. Because any disclosed computational pathway can, in principle, be selected or reconstructed after observing a desired output, ECF cannot distinguish between a genuinely forward, input-driven process and one that has been retrospectively constructed to match a pre-chosen result. However, if the experimental and plotting processes are disclosed as expected, evaluators can often use the provided computational artifacts, together with their own input data, to perform targeted tests that probe whether the forward process behaves consistently. In this sense, ECF does not guarantee irreversibility, but it provides the necessary evidentiary structure that allows evaluators to investigate whether a reported visual claim could plausibly arise from a legitimate forward computation.

Does ECF eliminate selective reporting?

The appropriateness of selective reporting depends fundamentally on the strength of the scientific claim being made. For example, a scientific claim such as "we can produce a desired output" asserts only the existence of at least one successful outcome. In such a case, presenting a single valid instance is sufficient, since the claim does not require frequency, stability, or typicality. By contrast, a stronger claim such as "we can reliably produce this output with a reported success rate or frequency" asserts a distributional property over many possible outcomes. Here, selective reporting becomes an evidentiary failure: omitting unsuccessful runs would undermine the validity of the claim itself.

When authors make the stronger type of claim, ECF provides the evidentiary visibility necessary for evaluators to detect selective reporting. By requiring disclosure of the relevant experimental or analytical processes, ECF enables reviewers to assess whether the reported visual claims reflect a stable pattern rather than a selectively filtered subset of instances. In this way, ECF does not eliminate selective reporting, but it supports a more informed and calibrated evaluation of the confidence that can be placed in the scientific claim.

A Peer-reviewed EOP/ECF-compliant Software Case Study

Here, we present a fully implemented example of EOP/ECF-compliant research software that has already passed conventional peer review (6). This example serves three purposes:

- to demonstrate that the principles of EOP are not aspirational but can be operationalized in practice;
- to illustrate how ECF provides a transparent, traceable linkage between software artifacts and the scientific claims they support; and
- to distill practical insights gained during development that may inform future EOP/ECF-compliant software.

Case Selection and Project Background

We report this case because it represents a recent peer-reviewed study in which we applied the EOP/ECF principles alongside the development of the research itself. Because all visual claims in this project arise entirely from a software-mediated procedure, the system provides a particularly transparent setting in which the evidentiary pathways can be understood, inspected, and manually tested. The conceptual advancement of EOP/ECF and the progression of this five-year research project were developed almost simultaneously, allowing us to directly experience the practical challenges encountered during implementation. These challenges also motivated the inclusion of additional stakeholders in the discussion and informed the reflections articulated in both the main text and the supplementary materials.

The published article investigates how distinct topological primitives influence the performance of artificial neural networks, contributing to the theoretical foundations of machine learning. It focuses on three-node network motifs and examines how they shape representational capacity and numerical stability, and, consequently, how they affect learning behavior, learning performance, and noise tolerance in artificial neural networks. The study comprises five components: static motif analysis, dynamic motif analysis, fixed-network analysis, evolved-network analysis, and real-world demonstrations.

Software Architecture and Its Interpretation Under ECF

Excluding minor annotation files and auxiliary folders automatically generated by the GitHub ecosystem, the research software is organized into three top-level directories ("effect", "practice", and "works") together with a project-level "README.md" file. The "effect" and "practice" directories contain scripts implementing the core methodological components of the project, while the "works" directory includes the full pipeline for running experiments and generating all visual outputs. Detailed descriptions of the main files are provided in Tab. S1.

Tab. S1 | Research software architecture of the reported case. For this code repository, the core codes are organized into the "effect" and "practice" folders, while the experiment, packaging, and plotting scripts are housed in the "works" folder. This code repository is openly accessible on GitHub.

code repository	
├── effect	Source codes of 3-node motif effect experiments.
│ ├── __init__.py	Exhibition of class and method calls, as well as the implementation of monitor class.
│ ├── networks.py	Classes related to the structure and training/testing workflow of three-node network motifs.
│ ├── operations.py	Motif database generation and output calculation.
│ ├── robustness.py	Statistical methods related to numerical stability.
│ ├── similarity.py	Statistical methods related to representational capacity.
├── practice	Source codes of neuroevolution experiments.
│ ├── __init__.py	Exhibition of class and method calls.
│ ├── agent.py	Trained agent (evolved neural network) classes.
│ ├── evolve.py	Neuroevolution method variation.
│ ├── motif.py	Motif-related operations and statistics.
│ ├── noise.py	Noise generation modules.
│ ├── task.py	Neuroevolution task modules.
├── works	Experiment and visualization modules of this work.
│ ├── confs	Configuration folder of neuroevolution tasks.
│ ├── data	Painting data folder of all the experiments.
│ ├── raw	Raw data folder of all the experiments.
│ ├── show	Painted figure folder of all the experiments.
│ ├── temp	Temp folder to temporarily save all the figures in Video S1-S8.
│ ├── __init__.py	Encapsulation of data storage and retrieval operations.
│ ├── run_1_tasks.py	Running all experiments for this work.
│ ├── run_2_packs.py	Packaging all the presented data from the experimental results.
│ ├── show_main.py	Figure plotting (in the main text) from the generated data.
│ ├── show_supp.py	Figure plotting (in the supporting materials) from the generated data.
│ ├── show_video.py	Video creation (in the supporting materials) from the generated data.
├── LICENSE	License of this code repository (GPL-3.0 license).
├── README.md	Description document of this code repository.
├── requirements.txt	Basic library requirements of this code repository.

Although the software does not exhibit any specific architectural patterns compared with conventional research software, its file organization is deliberately structured to facilitate evidentiary inspection. The core methodological components are isolated in the "effect" and "practice" directories, enabling

direct reuse or extension, while all evidence-bearing procedures are consolidated in the "works" directory. This separation is designed to make the evidentiary pathway transparent and easy to audit, particularly for evaluators who wish to verify visual claims.

In this code repository, all experimental designs and parameter settings are explicitly specified (see "run_1_tasks.py" and the "confs" folder). By executing "run_1_tasks.py", users can reproduce identical or statistically similar output data in the "raw" folder within a matter of months. This output data is then packaged by "run_2_packs.py" into a compact set of plotted data files (i.e., plotted data), which are stored in the "data" folder. All figures and videos in both the main text and the supplementary materials are generated exclusively from the data folder using "show_main.py", "show_supp.py", and "show_video.py", ensuring that the presentation contains no anthropogenic filtering, rearrangement, or adjustment. The resulting visual claims are stored in the "show" folder. The evidence chain execution pipeline corresponding to the above execution sequence is illustrated in Fig. S1.

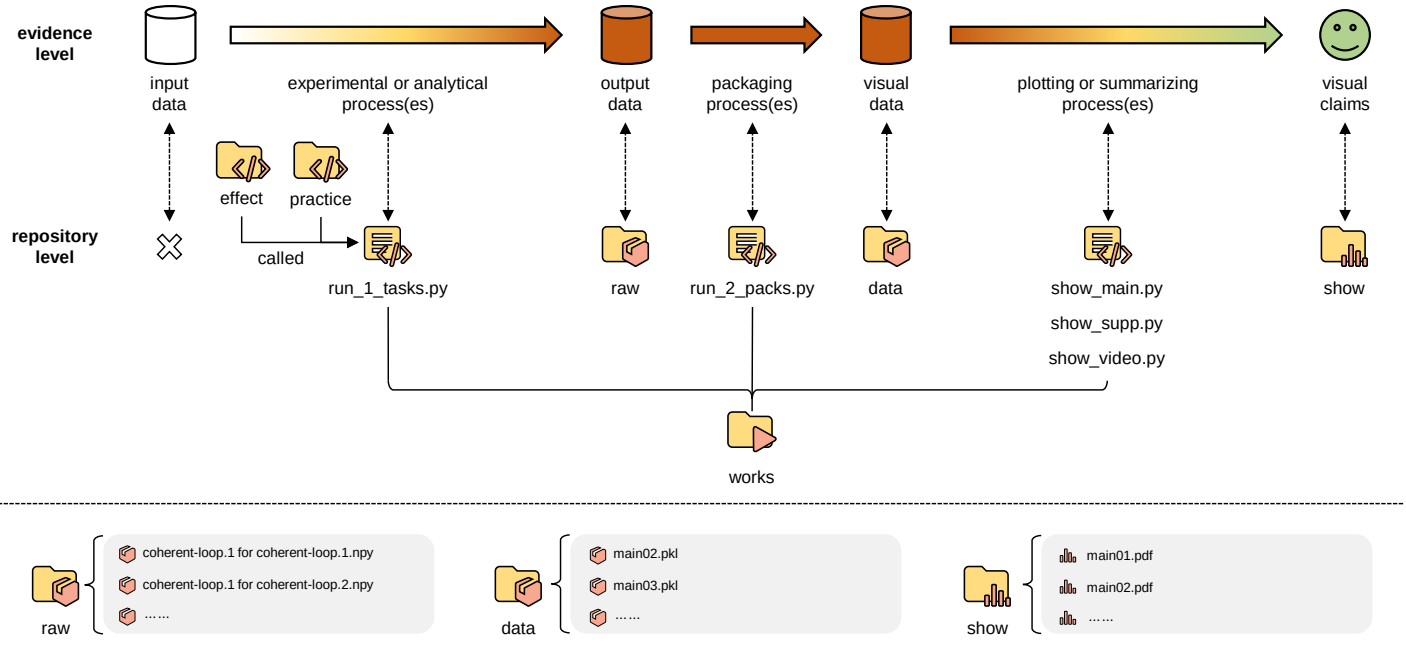

Fig. S1 | The relationship between code repository structure and the evidence chain. To repeat the entire evidence chain, users may simply execute the scripts from the "works" directory in the following order: (1) run_1_tasks.py; (2) run_2_packs.py; (3) show_main.py, show_supp.py, and show_video.py. The "effect" and "practice" folders represent methodology of this study. In other research contexts, these components may be realized not as source code but as distributable executables or network-based software access services. Such differences in representation do not, by themselves, undermine evidentiary sufficiency.

Significance of Intermediate Output Data

This project spans a five-year period, during which retaining intermediate output data proved essential for avoiding redundant reruns of computationally intensive tasks. When stored at an appropriately fine level of granularity, intermediate data not only prevent unnecessary recomputation but also facilitate additional analyses that were not part of the original experimental plan. Such data enable authors to perform supplementary investigations quickly without restarting the full experimental pipeline.

Because some scripts evolved across multiple versions throughout the five-year development cycle, we re-executed the entire pipeline on the KAUST IBEX high-performance computing cluster shortly before manuscript submission to ensure the internal consistency of all reported results. During this process, we found that reproducing the full workflow on a single consumer-grade laptop can require several months of computation, far exceeding the typical two-week window of a standard peer-review cycle.

Moreover, the value of intermediate data extends beyond peer review. Future readers may wish to perform independent analyses, recombine specific components, or integrate certain results into their own research. Access to curated intermediate outputs substantially reduces the effort required to conduct such follow-up work.

Despite these benefits, storing intermediate data introduces nontrivial practical considerations. When saved at a scientifically meaningful granularity, the total storage requirement can reach tens of gigabytes (i.e., output data; 91.4 gigabytes in our case), whereas journals typically request only compact source data files – often at the scale of tens of megabytes (i.e., plotted data; 71.5 megabytes in our case). Although this additional storage is beneficial for transparency, reproducibility, and extensibility, it nonetheless imposes extra costs for authors.

Additional Workload

The additional workload introduced by adopting an evidence-oriented organization of software and data varies substantially across disciplines, disclosure expectations, and project complexity. In the present case, the evidentiary structuring of the code repository did not generate additional burden on the software development side. Instead, the primary effort lay in deliberately organizing data and scripts in a manner that anticipates how different users (including peer reviewers and future readers) might wish to inspect, validate, or extend the computational results.

This form of organizational overhead is modest but intentional. It reflects a shift from viewing software solely as an implementation instrument toward managing it as an evidentiary asset. The effort is therefore not measured in additional lines of code or technical complexity, but in the conscious curation of data flow, intermediate outputs, and directory structure to support transparency, verifiability, and reuse. Looking forward, tailored AI agents designed to assist with evidentiary organization may further reduce the human effort required for managing or restructuring research software, particularly in large or long-running research projects.

Stage-wise Software Development Suggestions

Building upon the preceding discussion of our EOP/ECF-compliant software case, here we synthesize the practical insights gained during that multi-year development process and convert them into a set of stage-wise development suggestions. These development suggestions are not tied to any particular programming framework, software pattern, or architectural paradigm. Instead, they reflect the observation that the primary difficulty in developing EOP/ECF-compliant research software lies not in mastering particular tools, but in cultivating an evidence-oriented mindset. The development suggestions outlined below are deliberately lightweight and adaptable, enabling researchers from heterogeneous fields to adopt them incrementally without extensive re-engineering of existing workflows.

Initializing a Structured Software Archive

The foundational awareness required for applying evidence chains in research software development begins with the careful structuring of the software archive. A well-organized software archive not only improves the clarity and maintainability of the repository but also provides the basis for linking scientific claims to the specific files and operations that support them. It is generally advisable to adopt clear directory conventions, such as those illustrated in Fig. S2, which offer a template suitable for most research projects, though the structure may be further adapted depending on the actual disclosure needs of a given research project. The key principle is to ensure that all elements critical for reproducibility and evidentiary traceability are explicitly captured within the archive, making the repository self-contained and interpretable to others, including collaborators and reviewers.

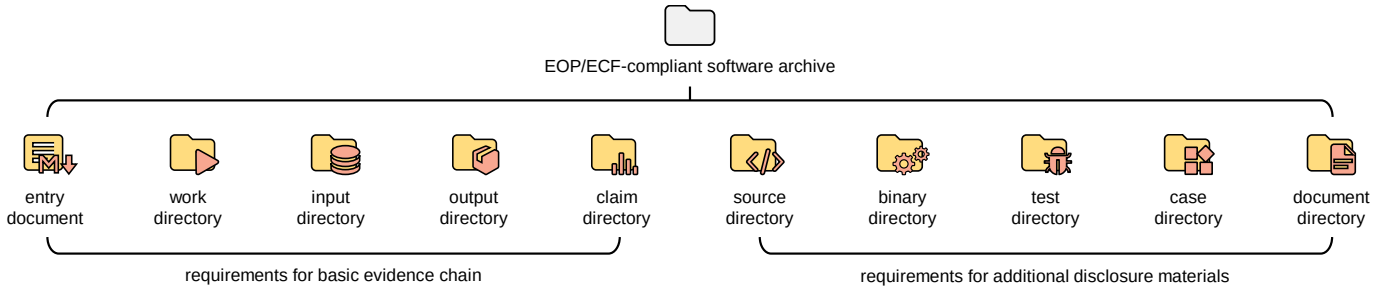

Fig. S2 | Suggested directory structure of the EOP/ECF-compliant software archive. Among these, the first five directories correspond to the basic disclosure for reconstructing the evidence chains, whereas the remaining directories represent additional materials depending on research contexts and negotiated expectations. The existence of a directory depends on the specific disclosure contexts.

The intended roles of each artifact illustrated in Fig. S2 are detailed below, and their specific naming conventions may be adapted as needed.

- **entry document:** The entry document serves as the starting point of the entire archive. It should specify the scope of the associated publication, describe the directory structure, and outline the execution environment and the intended run order of the pipeline. In addition, it provides readers and reviewers with a high-level map of how the archive is organized and how the evidence chain can be executed end-to-end.
- **work directory:** The work directory contains all execution pipelines required to produce the results reported in the academic publication. When multiple pipelines are involved, it is advisable to number them according to their intended execution order. A typical sequence begins with running experimental tasks to generate output data, followed by packaging or transforming these outputs into plotted data that correspond to specified visual claims, and finally rendering figures or tables from the plotted data. In some projects, however, the evidentiary workflow relies on interactive software interfaces rather than fully scriptable operations. When this occurs, the work directory may instead include a curated set of screenshots documenting the relevant software actions.
- **input directory:** The input directory is optional and appears only when the project depends on external data not generated by the software itself. In some projects, it may instead contain configuration files, presets, or input specifications required to initialize particular tasks. Its role is to make explicit any non-software-generated components in the evidence chain. When input data have already been deposited in public repositories, this directory may instead provide persistent links, access instructions, and any applicable permission or credential requirements.
- **output directory:** The output directory stores all generated data, including intermediate output data, final output data, and the plotted data used to produce visual claims. Multiple subdirectories may be included to separate data by experiment type, seed, task variant, or stage in the pipeline. Its function is to preserve a traceable record of all computationally derived evidence.
- **claim directory:** The claim directory stores the rendered and finalized visual claims of the academic publication: figures, tables, and, where applicable, videos. These files should be entirely derivable from the plotted data stored in the output directory and reflect the final evidentiary products disclosed to readers.
- **source directory:** The source directory contains project-specific algorithms, data structures, or core methodological implementations used to perform the analyses. Code scripts in this directory are typically reusable and independent of the evidence-bearing scripts.
- **binary directory:** The binary directory contains executable programs or compiled modules that implement project-specific logic in compiled form. Whether such binaries are disclosed depends on the negotiated expectations among the relevant stakeholders. When provided, they must correspond exactly to the functionalities invoked by the run directory.
- **test directory:** The test directory includes essential tests (e.g., unit tests, integration tests, or minimal validation routines) that demonstrate the correctness of the source code or binary executables. By confirming that the software behaves as intended, it substantiates confidence in the customized methods and data structures that underpin the evidence chains.
- **case directory:** The case directory offers minimal working examples illustrating how to invoke the source code or binary executables. Because textual descriptions alone may not always enable developers or users to form precise operational understanding, these cases help constrain user expectations and provide a quick way to explore or probe the behavior of the software.

- **document directory:** The document directory contains extended documentation related to the archive. Examples include detailed methodology descriptions, community-related statements, metadata files associated with external documentation websites (e.g., Read the Docs), or additional explanatory materials that are not required for executing the evidence chains but are useful for contextual understanding.

Managing Evolving Software Components and Data Elements

Developing research software becomes substantially more challenging as a project progresses, because both the software components and the data elements they produce inevitably evolve. This difficulty is particularly pronounced in multi-year research projects that involve ongoing software development: researchers may refine or replace specific algorithmic components, discover bugs that require re-execution of earlier computational tasks, adjust parameter configurations to obtain clearer or more representative results, or incorporate additional analyses to support emerging scientific claims. In parallel, visual claims produced at a particular stage may later need to be reused, extended, or adapted for other related projects. These realities make version control, data management, and evidentiary consistency considerably more complex than in conventional short-lived research projects or in non-research software development efforts with stable and well-defined expectations. This complexity also contributes directly to why research software often lacks evidentiary sufficiency, and we consider it to be among the major obstacles to developing EOP/ECF-compliant software.

From a practical standpoint, managing evolving software and data also depends on the technical proficiency of the research team and the degree of variability inherent to the research project. When researchers are confident in their software engineering practices and have a clear understanding of the eventual computational goals, it is feasible to organize the project directly following the directory structure illustrated in Fig. S2 and to incrementally construct the complete research software from the outset. Additionally, utilizing a version control system, such as Git, to manage iterative updates of software components will help ensure traceability throughout the evolving process. However, based on our experience with the EOP/ECF-compliant software case presented earlier and with ongoing EOP/ECF-compliant software efforts in other projects, a more flexible and scalable approach is often preferable: specifically, packaging each computational task as an independent module and maintaining a short accompanying document that records exactly what that computational task performs. These modular units can later be consolidated according to the needs of the final manuscript. When adopting such a modular approach, it is advisable to re-execute the entire pipeline after consolidation to ensure internal consistency across all computational artifacts in the overall evidence chain.

Lightweighting Evidence Chains and Improving Their Inspectability

Even when a software archive is well structured and its evolution is carefully managed, a persistent challenge remains: third-party evaluators cannot realistically inspect every script, trace every data transformation, or re-execute the full computational pipeline underlying a research project. Constrained by limited time, constrained computational resources, and incomplete familiarity with project-specific source code or binary executables, evaluators must assess evidentiary sufficiency under conditions far more restrictive than those faced by the original authors. Hence, the evaluability of research software depends not only on the evidentiary sufficiency provided by developers, but also on how effectively the evidence chain is presented in a form that is both lightweight and directly inspectable.

Several lightweighting practices can make indirect interactions between developers and third-party evaluators more effective. One useful strategy is to remove computational artifacts that do not contribute to the reported visual claims, such as deprecated analyses and their associated output data. Reducing such nonessential artifacts helps evaluators focus on the evidentiary components that actually support the publication. In addition, clearly distinguishing the primary evidence chain from auxiliary branches enables evaluators to prioritize the most consequential parts of the software archive (e.g., show_main.py and show_supp.py in Tab. S1). This separation not only reduces cognitive load but also accelerates the establishment of a shared understanding between developers and evaluators regarding how the reported visual claims are derived.

Beyond lightweighting evidence chains, it is equally important to provide mechanisms that enhance the direct inspectability of key computational steps. A practical way to achieve this is to expose intermediate outputs at a granularity that allows evaluators to confirm the correctness of specific transformations without reconstructing the entire computational pipeline. For such inspection to be feasible, each computational unit must complete within a reasonable amount of time; otherwise, reviewers may be unable to validate critical steps within the typical review period (often on the order of two weeks). At the same time, the granularity of these intermediate checkpoints must be carefully chosen: overly coarse checkpoints obscure the logic of the workflow and prevent meaningful validation, whereas overly fine-grained outputs introduce unnecessary clutter and hinder comprehension. Appropriate checkpointing thus forms the foundation of direct third-party verification, allowing evaluators to probe the computation at points they consider most informative while still maintaining a manageable evidentiary surface.

Negotiating Staged Supplemental Disclosure During Peer Review

While we recommend disclosing all feasible computational artifacts at the time of initial manuscript submission, certain exceptional circumstances may warrant staged supplemental disclosure during the peer review process. Such circumstances primarily arise when full and immediate disclosure of specific evidence-bearing components may introduce non-scientific risks, particularly in cases where authors and reviewers occupy potentially competing scientific trajectories or where latent conflicts of interest may exist (7–12). In these situations, prematurely releasing computational artifacts that directly encode core methodological innovations, key algorithmic insights, or domain-specific data-processing strategies may inadvertently enable unintended downstream use prior to publication, thereby creating asymmetric incentives not aligned with the norms and expectations of peer review.

To mitigate such asymmetric incentives while still preserving the evaluability of the scientific claims, a practical approach is to mark sensitive computational artifacts within the research software archive, explain to editors why their premature disclosure may raise conflicts of interest, and request that these materials be made accessible to reviewers only during the later stages of peer review. This form of editor-side escrow preserves evidentiary accountability – ensuring that the artifacts exist and can be verified at a later stage – without prematurely exposing them to potentially competing scientific trajectories. Authors adopting this approach remain responsible for ensuring that the provided materials are complete, stable, and consistent with the claims in the manuscript, and for releasing them to reviewers if and when the editor determines that direct access is required for peer evaluation.

Notably, no broad consensus currently exists within the community regarding the appropriateness of such staged disclosure practices. Their applicability may depend on multiple factors, including but not limited to editorial confidence in the authors, the willingness of reviewers to engage constructively when partial access is provided, and the norms of the relevant research community. While such practices are not the primary focus of the present initiative, broader community-level discussions may ultimately be beneficial for clarifying their scope, limitations, and appropriate use cases.

Supporting Practical Adoption Through Case Studies and Behavioral Tests

Once a manuscript is published and its accompanying EOP/ECF-compliant research software becomes accessible, a new challenge may emerge. Authors often hope that their work will receive broader attention and that more users, including those outside the immediate research community,

will engage with the software, explore its capabilities, or incorporate it into their own scientific activities. However, many of these downstream users may not read the academic publication in detail or may approach the software with expectations shaped by their own backgrounds, which can lead to mismatches between the software's intended evidentiary role and how it is actually used in practice. In such cases, additional strategies are needed to help align user understanding with the expectations embedded in the EOP/ECF-compliant research software.

Two practical strategies can help promote this alignment. First, providing concise case studies offers users concrete demonstrations of how the software is intended to be invoked and what behaviors are expected. Second, offering lightweight testing mechanisms enables users to quickly verify that their usage aligns with the intended operational patterns, even if they have not fully absorbed the operational scope articulated in the publication. Together, these strategies lower the barrier for correct engagement and help ensure that broader adoption of the software remains consistent with the evidentiary intentions of the original authors.

Evidence Chain Differentiation across Research Contexts

Here, we examine how disclosure expectations for research software vary across research contexts, and how such variation mediates the differentiation of evidence chains.

Software Usage Variation in Scientific Activities

The evidentiary sufficiency expectations of research software vary substantially across academic fields since disciplinary norms impose structurally different expectations on what must be disclosed for scientific claims to be verifiable. In the early stages of developing this initiative, we surveyed several representative academic fields spanning the experimental-to-computational spectrum and conducted extended discussions with domain experts on the disclosure practices they consider necessary for evidentiary sufficiency. Although the concrete emphases differed widely across fields, we observed that these practices nonetheless converged on a more abstract, field-agnostic structure.

To make the field-agnostic structure more explicit, we first examine how research software is actually used within scientific activities. Our cross-field discussions revealed that what ultimately drives evidentiary differences is not the software's external form (whether it appears as a script, package, workflow, or service) but the computational role it plays in producing, transforming, or conditioning the evidentiary basis of a scientific claim. Accordingly, differentiating evidence chains requires distinguishing the ways in which software operations are invoked, combined, and initialized.

Although academic fields adopt research software for markedly different purposes, the recurrent modes of usage discussed by stakeholders converged on three relatively independent dimensions (as organized in Tab. S2). These dimensions provide a functional decomposition of software usage: they describe the structural complexity of computational invocation, the degree of methodological novelty embodied in computational operations, and the difficulty of initializing those operations before execution. Taken together, they define the primary ways in which research software can influence the evidentiary foundations of scientific claims, i.e., visual claims including figures, tables, and statistical quantities.

Tab. S2 | Three dimensions of software usage. Research software takes diverse forms (13), yet from a usage perspective these forms all instantiate the same underlying notion of a *computational operation*. To ensure conceptual clarity, we adopt this abstract notion to describe usage variation and to avoid interpretation ambiguities arising from software form differences. In addition, hyperparameters are methodological settings specified prior to execution, whereas parameters are quantities whose values are inferred or initialized from predetermined data.

dimension	level	explanation
invocation complexity	isolated unit	use of a single computational operation without dependencies on other computational operations
	sequential pipeline	use of multiple computational operations in a fixed, linear pipeline order where each step consumes the output of the previous one
	multi-branch workflow	use of workflows involving branching, merging, or multiple dependent computational operation paths, forming a non-linear computation structure
methodological novelty	existing computational operation	use of established computational operation(s) whose methodological design and computational objectives are well characterized and widely adopted
	modified computational operation	use of computational operation(s) whose methodological design has been partially altered while retaining the same computational objectives as those of the established computational operation(s)
	novel computational operation	use of computational operation(s) that introduce new computational objectives or a fundamentally different methodological design relative to the established computational operation(s)
initialization difficulty	no initialization required	use of computational operation(s) that can be executed directly after installation without any pre-execution procedures
	hyperparameter initialization	use of computational operation(s) that require hyperparameter declaration prior to execution, including explicit methodological hyperparameters and implicit initialization hyperparameters (e.g., random seeds)
	parameter initialization	use of computational operation(s) that require both hyperparameter declaration and parameter initialization (e.g., model training or fine-tuning) prior to execution

Although these dimensions are analytically separated for clarity, they are not entirely independent in practical implementation. For instance, higher invocation complexity often requires higher initialization difficulty, and methodological novelty may influence both invocation complexity and initialization difficulty. Nevertheless, treating the dimensions as conceptually distinct is useful for evidentiary analysis because each dimension introduces a different mode of uncertainty into the evidence chain: invocation complexity affects traceability, methodological novelty affects justification, and initialization difficulty affects reproducibility. Their interactions therefore reinforce rather than diminish the utility of distinguishing them when assessing the evidentiary role of research software.

Software Disclosure Expectations Driven by Software Usage

From a disciplinary perspective, researchers use research software for a wide range of methodological purposes. However, these purposes can generally be abstracted into a small number of concrete objectives. These objectives capture why the software is introduced into a scientific workflow and can be roughly grouped into four categories: (1) analyzing and visualizing newly produced experimental data; (2) establishing a computational method to solve a specific problem; (3) demonstrating the superiority of a new computational method; (4) evaluating computational methods using new and/or widely adopted data. Although the specific methods vary across fields, these high-level usage objectives are sufficiently broad to organize the corresponding evidentiary expectations.

Once the intended use of the software is identified, the three dimensions of usage outlined in Tab. S2 define how the software influences the scientific claim and shapes its evidentiary foundation. A central expectation is that when a scientific claim relies on a computational operation from software, that operation and its associated data context become part of the evidence and should be disclosed. Accordingly, disclosure expectations arise from the interaction between the software's intended use (its software usage objective) and how it is employed (its invocation complexity, methodological novelty, and initialization difficulty, as outlined in Tab. S2). Fig. S3 illustrates how these disclosure expectations manifest across the four usage objectives, emphasizing both the evidentiary role of research software and the risks introduced by insufficient disclosure. These examples highlight that disclosure expectations are inherently task-dependent, shaped by the software's role within the scientific activity. Although the specific consequences of insufficient disclosure of certain computational artifacts on visual claims require further analysis, the most significant impacts generally arise from incomplete or unclear disclosure of the computational artifacts directly leading to the visual claims.

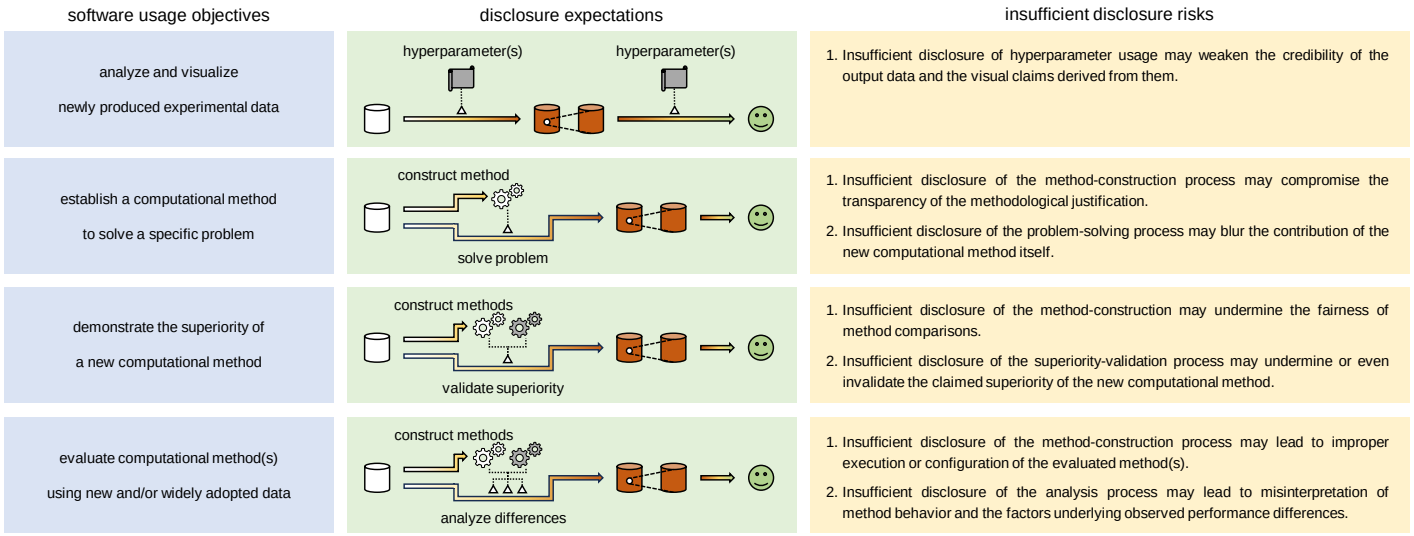

Fig. S3 | Software disclosure expectations vary across four typical software usages. The ways in which software is used in scientific activities define both its potential influence on scientific practice and the risks introduced by insufficient disclosure. The icons in the software disclosure expectations is defined in Fig. 2. In addition, the gear icons represent either established or newly developed computational methods. The triangle icons indicate how computational methods or other data elements interact with or contribute to the computational processes.

Claim-contingent Adjustments to Software Disclosure

At a finer level of granularity, the appropriate level of software disclosure also depends on the specific scientific claim that the software is used to support. Different scientific claim types rely on different portions of the evidence chain, and therefore the evidentiary components that must be disclosed can vary across research contexts.

This claim-contingent disclosure variation is particularly salient in the second usage objective in Fig. S3, where the purpose is to establish a computational method to solve a specific problem. When the central claim concerns whether the method can in fact solve the problem, the evidentiary burden is focused on demonstrating that the instantiated method performs the computational operations necessary to produce a valid solution. In such cases, disclosing the complete construction of the new method may not be strictly required, provided that the constructed computational method is accessible, unambiguous, and sufficient for demonstrating the claimed performance.

In contrast, when the claim asserts that a new computational method is superior to existing alternatives, the evidentiary boundary expands (i.e., the third usage objective in Fig. S3). For this objective, the construction details of the new method becomes part of the evidence chain because the fairness and interpretability of the comparison depend on understanding how the method differs from prior approaches and how those differences influence performance. In this setting, insufficient disclosure of the method-construction process may weaken to judge whether the comparison is fair.

Thus, the scope of software disclosure should be adjusted according to the evidentiary relevance of specific methodological details to the claim. Researchers are not required to disclose all components of a method simply because they exist, nor can they omit components that materially influence the claim under evaluation. By determining which artifacts lie on the evidence chain required to substantiate a specific claim, authors can calibrate the disclosure level in a manner that is both sufficient and proportional.

Evidentiary Verifiability Decoupling and Supplementary Disclosure under External Disclosure Barriers

Here, we focus on external disclosure barriers on research software and discuss their legitimacy within contemporary research ecosystems. Building on this analysis, we examine whether alternative forms of disclosure can be adopted to maintain evidentiary sufficiency when full transparency is constrained by external considerations.

Typical External Disclosure Barriers

In contemporary scientific activities, research software disclosure often encounters external barriers shaped by a combination of scholarly and non-scholarly considerations. These barriers have become increasingly visible in recent years, reflecting a growing clarity across the broader research ecosystem regarding the role of academic publication within scientific activities. From a pragmatic perspective, academic publication is increasingly regarded as only one component of research output, while expectations concerning the societal, commercial, or strategic value of research outcomes are shaped by actors operating beyond the publication process. Accordingly, academic publication is increasingly treated as a rigor-anchored channel for communicating and validating research outcomes, rather than as a venue for unrestricted dissemination of all associated research artifacts. The verification of rigor underlying the scientific claims articulated in a manuscript may therefore operate orthogonally to, and in some cases come into tension with (14), these external disclosure barriers during the publication process.

Based on recent publication practices, three recurring types of external disclosure barriers can be identified.

- **Proprietary licensing:** This barrier imposes legally binding constraints rooted in contractual frameworks. It arises when computational artifacts circulate across organizational and institutional boundaries under licensing agreements that restrict access or redistribution. Violations of such agreements may expose authors and their institutions to legal liability.
- **Commercial interests:** This barrier constrains the disclosure of information that confers competitive advantage. It emerges when computational artifacts encode proprietary expertise, optimization strategies, or commercially sensitive design decisions whose early or detailed exposure could erode their market value.
- **Security concerns:** This barrier reflects constraints grounded in risk assessment and harm prevention. It arises when computational artifacts enable capabilities that may be repurposed beyond benign research use, including dual-use functions or sensitive implementation details.

Implications of External Disclosure Barriers for Evidence Chain Formalization

ECF presumes that input data can be directly and coherently transformed into output data through experimental or analytical processes. A subset of these outputs is then rendered as visual data and, through the subsequent plotting or summarizing processes, gives rise to the visual claims. External disclosure barriers challenge this presumption not by uniformly restricting all computational artifacts, but by selectively constraining access to specific software components or data elements, thereby disrupting the default connectivity of the evidence chain (i.e., full accessibility of the computational artifacts involved). Understanding how these barriers reshape ECF is essential for assessing whether evidentiary sufficiency can be maintained when full access to computational artifacts is not feasible.

Although external disclosure barriers arise from distinct considerations, their implications for ECF tend to converge, consistently influencing the connectivity of the evidence chain in a small number of critical locations (Fig. S4). In particular, these barriers affect the formalization of the evidence chain at three interconnected points. They may constrain the accessibility of input data, which anchors experimental or analytical processes; may limit access to certain software components within these processes, thereby restricting the ability to fully instantiate or interrogate the corresponding computational operations; and may influence whether portions of the output data are recorded, especially when such outputs can be meaningfully traced back to restricted input data and thus inherit the disclosure constraints associated with the input data.

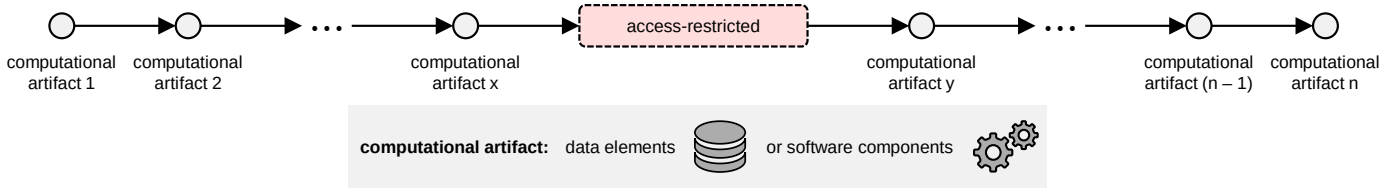

Fig. S4 | Evidence chain with an access-restricted computational artifact. Here we illustrate an abstract evidence chain. Due to external disclosure barriers, a computational artifact cannot be directly linked through a fully accessible disclosure mechanism, thereby disrupting the connectivity of the evidence chain under default assumptions. In practice, such a computational artifact may also constitute the starting point of the evidence chain, for example as restricted input data.

It is important to reiterate that, while the presence of external disclosure barriers may restrict direct access to specific computational artifacts, it does not imply that any element of the evidence chain can be entirely exempted from disclosure. Rather, such barriers should motivate the reconfiguration of evidentiary disclosure at the computational level, rather than serve as a rationale for resisting third-party verification. Clarifying this distinction is essential to prevent external disclosure barriers from being invoked to legitimize evidentiary gaps, as disclosure constraints and verification demands need not be in fundamental opposition. As a supplementary consideration, when external disclosure barriers are sufficiently broad to render any form of verifiable disclosure infeasible at the computational level, it becomes necessary to reconsider whether academic publication remains an appropriate vehicle for dissemination. In such cases, alternative channels that do not rely on third-party verification or academic endorsement may offer a more flexible and context-appropriate option.

Decoupling of Evidence Verifiability from External Disclosure Barriers

External disclosure barriers can pose practical obstacles to the accessibility of computational artifacts. Yet, viewed through the structure of the evidence chain, such barriers do not necessarily undermine the verifiability of a scientific claim. This is because the fundamental evidentiary requirement is whether a computational artifact exists and whether it fulfills its intended functional role within the evidence chain.

In practice, we observe that such requirement manifests differently across types of computational artifacts. Software components often admit substantial flexibility in how access is provided without altering their evidentiary role, as their contribution to the evidence chain is primarily functional. In comparison, the evidentiary role of data elements is intrinsically tied to the data themselves, making their evidentiary function closely coupled to direct accessibility.

Accordingly, the scope of decoupling enabled by ECF is largely confined to software components. Specifically, beyond the two types of processes required by the basic evidence chain (Fig. 2), additional software components may be exposed in different forms without necessarily compromising evidentiary sufficiency (Fig. S5). So long as these components remain testable in relation to their evidentiary function, variations in disclosure primarily affect the modality of access rather than the role those components play within the evidence chain.

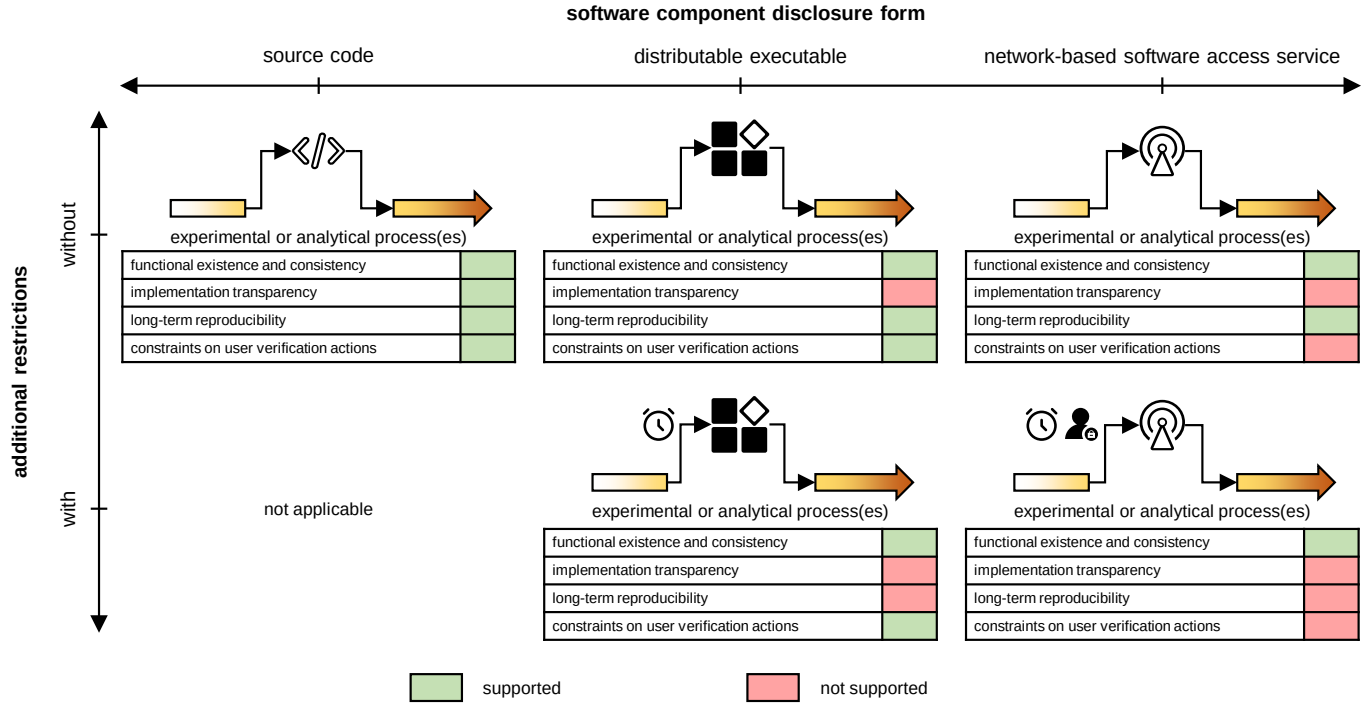

Fig. S5 | Evidentiary sufficiency and other software properties under different disclosure forms. When the connectivity of the evidence chain is preserved, the basic condition for evidentiary sufficiency (i.e., functional existence and consistency) is satisfied. Different disclosure forms of software components may differentially affect other software properties, necessitating case-specific adjustment and calibration.

Supplementary Disclosure Strategies under External Disclosure Barriers

While decoupling constitutes a principled mechanism provided by ECF for preserving evidentiary verifiability under certain external disclosure barriers, its scope is inherently limited. It is most effective for software components whose evidentiary role is primarily functional (Fig. S5), but does not fully address scenarios in which access restrictions apply to data elements or to multiple interconnected artifacts along the evidence chain. As a result, in many practical settings, decoupling by itself may be insufficient to ensure evidentiary sufficiency.

Accordingly, when decoupling alone cannot sustain evidentiary sufficiency, it may become necessary to introduce supplementary disclosure mechanisms that enable the evidence chain as a whole to satisfy its fundamental evidentiary requirements, namely verifiable existence and functional consistency. Here we outline two supplementary disclosure strategies that can be adopted under such conditions (Fig. S6). These strategies are not mutually exclusive and may be combined as needed, i.e., Fig. S6A can be a part of Fig. S6B. Their necessity depends on whether the evidentiary sufficiency of the research software can be reasonably established and agreed upon among relevant stakeholders.

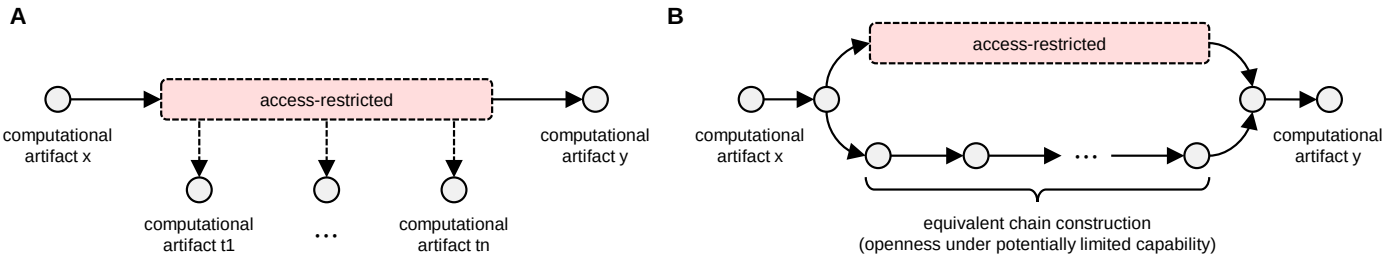

Fig. S6 | Two supplementary disclosure strategies. (A) Auditable intermediate data elements produced by the access-restricted computational artifact. (B) A sub-evidence chain that achieves restricted functional equivalence relative to the access-restricted computational artifact. Such functional restrictions may depend on the supported computational platform, limits on computational performance, or constraints on target objects.

Below, we provide a structured discussion of the two supplementary disclosure strategies introduced above. For each strategy, we clarify its underlying rationale in terms of how it restores or reinforces evidentiary traceability under access restrictions, and identify a set of key properties that are relevant for assessing whether the strategy can meaningfully contribute to evidentiary sufficiency.

- **Provide auditable intermediate data elements:** The central idea of this strategy is to re-anchor evidentiary traceability by disclosing selected properties, intermediate representations, transformations, or execution traces for verification. These auditable intermediate data elements act as observable anchors that attest to the existence, execution, and functional behavior of access-restricted artifacts within the evidence chain, and can be applied to both restricted data elements and software components. To meaningfully contribute to evidentiary sufficiency, such intermediates would need to satisfy a set of key properties, including (i) being verifiable, allowing independent parties to confirm their provenance, integrity, and linkage to the restricted artifact; (ii) being functionally specific, reflecting either distinctive properties of the restricted data or well-defined stages of the computation; and (iii) being non-sensitive and non-reversible, thereby imposing substantial barriers to the reconstruction of the access-restricted artifact by malicious actors. In practice, such intermediates may take the form of cryptographically signed execution traces on neutral platforms (e.g., Code Ocean), non-sensitive checkpoint outputs, or derived datasets that preserve structural or statistical properties relevant to subsequent accessible computational artifacts without exposing protected inputs, thereby enabling reviewers to assess consistency with reported conclusions even in the absence of direct access.

- **Construct the restricted functional equivalence relative chain:** The central idea of this strategy is to provide an alternative implementation that preserves the core computational behavior relevant to the scientific claim, while deliberately reducing performance optimizations, omitting sensitive design details, or restricting its scope of applicability. This strategy applies exclusively to software components, as data elements are generally not substitutable in this context, and their evidentiary role is intrinsically tied to direct access. To meaningfully contribute to evidentiary sufficiency, such an alternative implementation would need to satisfy a set of key conditions. First, it would need to be accessible in a manner sufficient for evidentiary assessment (typically through access to source code or an equivalently transparent reference implementation), allowing reviewers and other relevant stakeholders to execute and inspect the alternative implementation to evaluate its consistency with the reported methodological design. Second, the differences between the restricted implementation and the access-restricted artifact would need to be explicitly characterized, together with a clear justification of the intent and scope of the imposed restrictions. Third, the two implementations would need to admit synchronized verifiability, meaning that they can be evaluated within a shared experimental or analytical context to confirm functional consistency. In practice, restricted functional-equivalence implementations may take the form of simplified reference implementations, open-source versions with reduced efficiency or limited parameter ranges, or surrogate computational pipelines that reproduce the claimed behavior under controlled conditions. When appropriately scoped, such a restricted functional equivalence implementation allows reviewers to verify the soundness of the claimed computational logic and to assess whether the reported results depend on undisclosed mechanisms, even in the absence of full access to the original software component.

One-to-one Interviews of this Initiative

Motivation, Design, and Selection

As an integral component of promoting and evaluating the implementation of this initiative, we conducted a series of one-to-one interviews. This format was chosen to facilitate direct and detailed feedback, while also allowing the explanation of the initiative to be adaptively adjusted according to participants' backgrounds, with different aspects emphasized for different individuals. Given that the initiative is at an early stage of development, this approach also helped to minimize framing effects and premature judgments, thereby supporting more candid and exploratory feedback that is informative for subsequent refinement.

The design of the interviews reflected both the conceptual breadth of the initiative and its anticipated implementation context. Since the initiative spans a heterogeneous set of concepts, including research software practice, evidentiary standards, disclosure norms, and cross-disciplinary evaluation, participants representing different potential stakeholder roles were interviewed to examine how these ideas are interpreted upon first exposure. At the same time, as students and early-career researchers are expected to constitute a major group of prospective implementers, the interview cohort was intentionally concentrated at the student level.

Each interview was limited to a maximum duration of 30 minutes and focused on two closely related aims: introducing the core principles of the initiative to assess conceptual understanding and acceptance, and discussing practical aspects of software-related implementation – including disclosure scope and procedural implications – to evaluate perceived feasibility and willingness to adopt the proposed practices.

It is important to note that all participants had no prior knowledge of this initiative, nor of closely related software governance guidelines. This constraint ensured that responses reflected first-exposure understanding rather than pre-existing familiarity or alignment with similar initiatives.

Participant Information

Prior to each interview, participants were informed of several procedural details relevant to the documentation and disclosure of interview information. As part of the interview process, we collected and disclosed a summary table describing participant attributes, including name, affiliation, primary research field, and experience in software development. Participants were given the option to be listed either under their real names or anonymously. In cases of anonymous disclosure, name and affiliation was withheld.

The resulting participant information is presented in Tab. S3. This table is intended to provide transparency regarding the composition of the interview cohort and to clarify the diversity of backgrounds represented among interview participants.

Tab. S3 | Participant information for one-to-one interviews, ordered by interview time Participants' experience in software development was classified into three broad categories: no experience, familiar, and proficient. No experience refers to individuals who have no substantive familiarity with software development and have not participated in software development tasks. Familiar denotes participants who have received systematic instruction in software development or have undertaken practical development tasks. Proficient refers to participants who have majored in software engineering or closely related disciplines during their formal education and have received comprehensive, structured training in the field. In addition, interview schedule refers to the specific start time of each interview and is reported in UTC/GMT+3.

name	affiliation	primary research field	software development experience	interview schedule
Guanjin Qu	Tianjin University	applied mathematics	familiar	18:00, 4 December 2025
Mengge Wang	King Abdullah University of Science and Technology	cell biology	no experience	13:00, 11 December 2025
Minjing Su	King Abdullah University of Science and Technology	metagenomics	no experience	16:00, 11 December 2025
Xian Fu	BGI-Research	microbiology	no experience	13:00, 12 December 2025
Qidian Li	University of Oxford	inorganic chemistry	no experience	21:30, 13 December 2025
Fotios Kefalas	King Abdullah University of Science and Technology	molecular biology	no experience	16:45, 14 December 2025
Shirui Yan	Yale University	biochemistry	no experience	21:40, 14 December 2025
Yingzi Zhang	King Abdullah University of Science and Technology	molecular biology	no experience	14:00, 16 December 2025
Jin Su	Westlake University & Zhejiang University	computational biology	proficient	08:30, 19 January 2026

Supplementary References

1. C. S. Liew et al. Scientific workflows: moving across paradigms. *ACM Computing Surveys* **49** (2016), 1–39.
2. E. Snider et al. Retraction Note: Room-temperature superconductivity in a carbonaceous sulfur hydride. *Nature* **610** (2022), 804–804.
3. N. Dasenbrock-Gammon et al. Retraction Note: Evidence of near-ambient superconductivity in a N-doped lutetium hydride. *Nature* **624** (2023), 460–460.
4. M. Naddaf. ChatGPT generates fake data set to support hypothesis. *Nature* **623** (2023), 895.
5. N. Davydiuk et al. The rising danger of AI-generated images in nanomaterials science and what we can do about it. *Nature Nanotechnology* (2025), 1–4.
6. H. Zhang et al. Leveraging network motifs to improve artificial neural networks. *Nature Communications* (2025).
7. Z. Yan, Y. Peng, Y. Wu, and J. Di. Retraction: Controllable electrochemical synthesis of silver nanoparticles on indium-tin-oxide-coated glass. *ChemElectroChem* **2** (2015), 578–583.
8. N. M. Khan et al. Retracted: Potentiality of Neem (a zadirachta indica) powder in rheology modification of oil-in-water emulsion. *Journal of Food Process Engineering* **38** (2015), 190–196.
9. A. Anderson, Y. Devarajan, and B. Nagappan. Statement of Retraction: Effect of injection parameters on the reduction of NOx emission in neat bio-diesel fuelled diesel engine. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects* **40** (2018), 2771–2771.
10. G. Zhong, H. Deng, and J. Li. Retraction Note to: Chattering-free variable structure controller design via fractional calculus approach and its application. *Nonlinear Dynamics* **100** (2020).
11. S. Kumar et al. Retracted Article: Bismuth-based nanoparticles and nanocomposites: synthesis and applications. *RSC Advances* **14** (2024), 39523–39542.
12. D. Ghorbanzadeh et al. Retraction notice: Empirical nexus of corporate social responsibility, service quality, corporate reputation and brand preference: evidence from Iranian healthcare industry. *Journal of Health Organization and Management* **39** (2025), 147.
13. M. Gruenpeter et al. *Defining research software: a controversial discussion. Zenodo* (2021).
14. S. Wankowicz et al. *AlphaFold3 Transparency and Reproducibility. Zenodo* (2024).