Preserving analyses: where decomposition into graphs helps

Follow up on Research Objects

August 2026

Eric / Maxim /

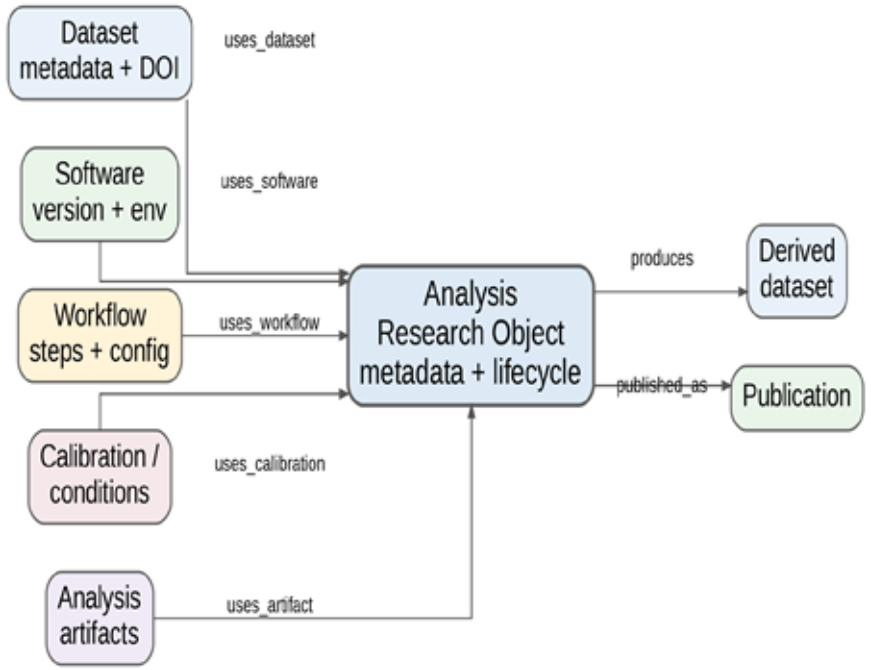
Several published analyses have been decomposed into graphs and the consistency of the graphs has been checked

Recall: Why represent an analysis as a graph?

A dataset can be described on its own. An analysis cannot.

- **An analysis is more than a dataset.** Data, software, workflows, configurations, calibrations, executions and derived results are all involved, and they are connected to each other.
- **A datacard describes a dataset.** It does not describe fully what was done to that dataset, with which code, under which settings, or which result came out.
- **The dependencies are the content of the analysis.** Which run list a measurement used, which library version produced a figure, and which model turned an observable into the published number are all relations between objects rather than properties of any one of them.

Research-object graph



The first deliverable is a datasheet for a dataset. The analysis objects are included because they are what make the datasetinterpretable and reusable.

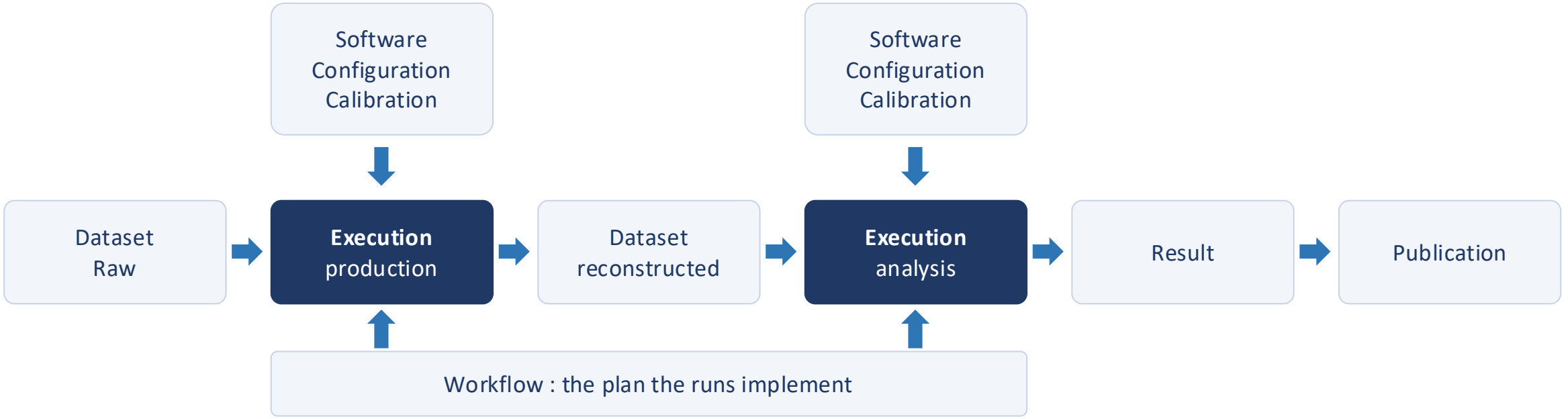
What has been tested

- Three published STAR analyses, decomposed into Research Object graphs with the help of an LLM
  - Sources: the analysis notes, the associated code repositories, and the references they cite

- What came out of it
  - Nine object classes were sufficient to decompose for all three analyses
  - Additional sources exposed intermediate objects and dependencies not visible in the note
  - **The graph produced an explicit list of unresolved dependencies and inconsistencies**

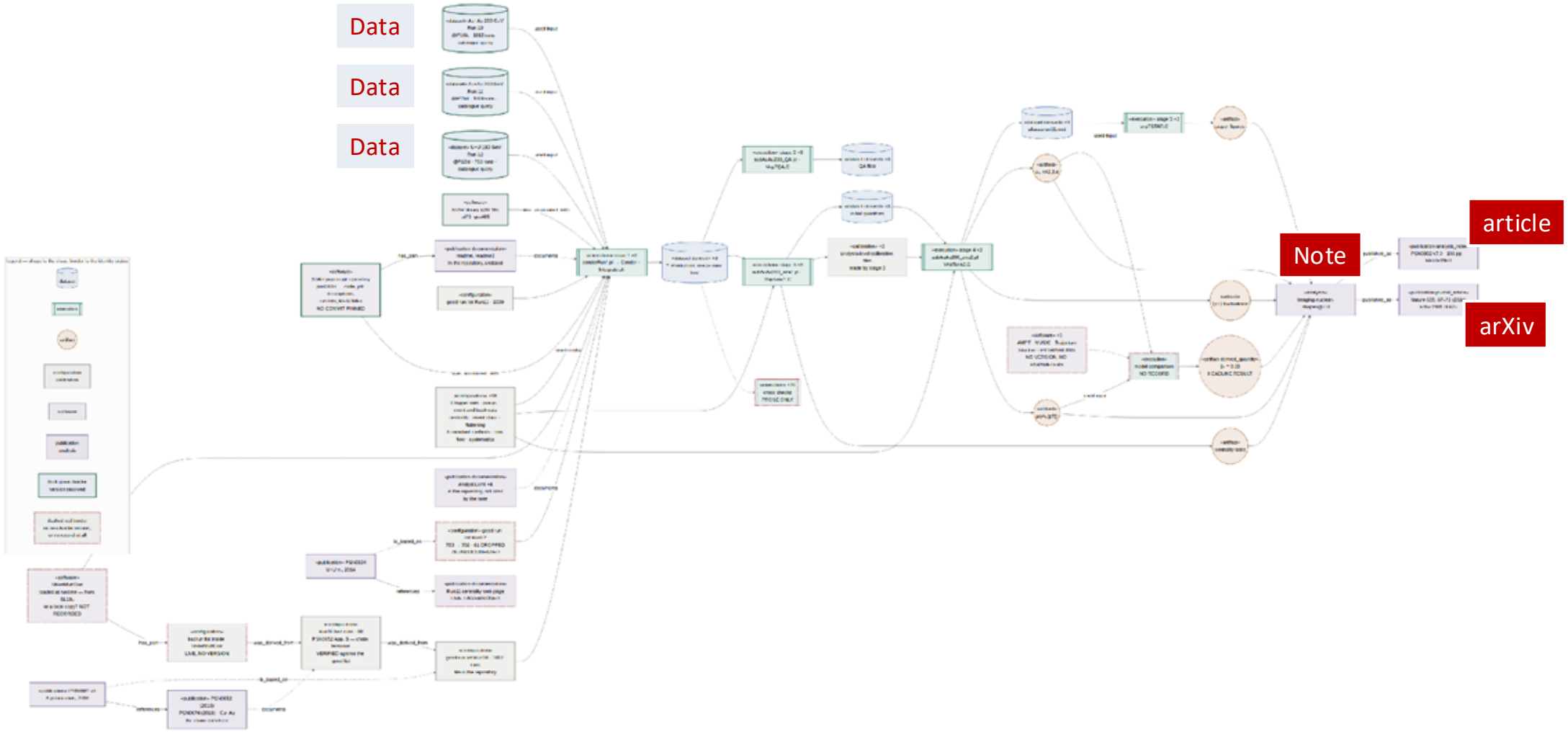
What a typical analysis decomposition looks like

Calibration is itself produced by an execution,



Nine classes (the boxes): *Configuration*, *Execution, Dataset, Software, Workflow, Artifact, Publication, Calibration, Analysis*
And **relations (the arrows)**: *used, was_generated_by, was_associated_with, was_derived_from, is_part_of, published_as*
Defined following some standards

One example of a graph



90 objects, 256 relations.

Examples of gaps revealed by the decomposition

The analysis can be described, but several dependencies cannot be pinned to the state actually used.

- Code state
  - The cited repository exists, but the actual publication code state is not pinned.
  - *Ex: The note cites GitHub, but does not identify the commit used to produce the published result.*
- Runtime dependency
  - An input is loaded at run time but its exact version is not preserved.
  - *Ex: a calibration is loaded at run time.*
- Shared input
  - A run list is inherited or maintained outside the analysis and has no dated version.
  - *Ex: The analysis uses a run list maintained elsewhere, but the exact run list used for the publication is not preserved*
- Documentation drift
  - The README describes a workflow that no longer matches the repository.
  - *Ex: The README describes one workflow, while the scripts in the repository now implement another*

The graph makes these gaps explicit by requiring each dependency to be represented and linked

Citation is not enough

A citation can point to the relevant code without identifying the code state that produced the result.

Example found by inspection

Published value

339

Value in cited code

338

No execution was needed to find the mismatch.

Issue:

Without a pinned commit or release, one cannot tell whether the code changed after the result was produced or whether the published value was produced differently.

One rule for every input

Every object an execution consumed must be resolvable to a fixed state.

- An identifier
  - A stable identifier, not only a file path.
  - *Example: a dataset or catalogue identifier rather than /data/…*
- A fixed, recorded reference
  - A commit hash, release, production identifier, manifest or snapshot, never “current”.
  - *The reference must identify the state actually used by the execution.*
- A retrievable copy
  - A location that survives, a personal directory or build area is not an archival location: it may disappear with the person or project.
  - *The object must still be available when the analysis is revisited years later.*

- **The important point is not one repository for everything, but a fixed, resolvable reference for every input.**

Shared inputs need their own authoritative version

An analysis is defined by shared inputs (used by several analyses) and analysis-specific choices. Both must be preserved.

Shared inputs

- Good/bad-run lists
- Calibration sets
- Standard selections
- Shared parameters
- Authoritative released version

Analysis-specific inputs

- Additional cuts
- Binning
- Sample-specific corrections
- Published with the analysis

Example: a shared bad-run list existed only as a live web page. Without a pinned version, an analysis cannot state which list it used.

Avoid private copies of shared inputs

Shared inputs should have one authoritative version that analyses reference.

The note and the code are complementary

Both are required. Neither can be reconstructed from the other.

- The note carries intent
  - Why this choice, how to interpret the result, what was checked, etc…
- The code carries behaviour
  - Exact values, order of operations, run-time inputs, and the effective selection.
- **Each should name the other**
  - The note names the exact code state it used; that state records which analyses it served, one package can serve many notes.

Record what the run actually used

The recorded values should be produced by the execution that produced the result

- **Generated by the run**
  - Cut values and binning
  - Run and event counts
  - Versions and checksums of every input
  - Effective selections actually applied
  - **Tables** and figures and data points

  Stored as a machine-readable record beside the result, not typed into the note.

- **Written by a person**
  - Why this choice was made
  - What the numbers mean
  - Caveats and open questions
  - The physics argument

Individual cuts → Effective selection actually applied

- `pT > 1 GeV`
- `|eta| < 1`
- `nHits >= 20`
- `DCA < 2 mm`

(pT > 1 GeV)
AND (|eta| < 1)
AND (nHits >= 20)
AND (DCA < 2 mm)
AND good_run
AND trigger_fired

Preservation practices

These can be implemented incrementally...

- Pin every input to a fixed state.
- Give shared inputs a version.
- Generate published numbers from the execution that produced them.
- Preserve the actual selection applied to the data, including how the individual cuts are combined.
- Review the note against the code state, not just the citations.

The graph is a review instrument.

Its value is not the diagram. It is the explicit list of dependencies, provenance gaps and inconsistencies that can be identified

Backup

Material cut from the main talk

Checks you can run on your own analysis

These checks found gaps in at least one of the three case studies.

- Does the environment reference carry a digest rather than a moving tag?
- Does any script actually produce the artifact the run script consumes?
- Is the input file list preserved as a file, with identifiers and checksums?
- Do the note's parameter values match the arguments the jobs actually passed?
- Does the documented recipe run, command by command, against the file tree?
- Are there files, scripts or directories the analysis depends on that are not under version control?

None of these checks requires running the analysis. They can be performed by inspection.

What decomposition exposed

Six recurring failure modes appeared across the three case studies.

- Citation without pinning
  - The reference resolves but not to the state that was used. From outside it looks like good practice.
- The mechanism fails, not the artifact
  - Moving branch pointers, run-time relative paths, an identifier stored only in a document's link layer.
- History that does not reach back
  - A package's recorded history can begin after the software release the analysis ran under.
- Hand-typed numbers drift
  - Tables transcribed once, early, from code that kept moving and the drift is invisible in the document alone.
- The execution record nobody kept
  - Job input manifests, environment. Without them everything downstream is provenance by inference.
- Documentation drift
  - READMEs naming a binary, an interface and a stage order that the repository no longer matches.