Scrubbing the Run: Replay-Native Provenance Visualization for Agent-Supervised Beamline Experiments

Doğa Gürsoy*
Advanced Photon Source, Argonne National Laboratory

Francesco De Carlo
Advanced Photon Source, Argonne National Laboratory

ABSTRACT

Autonomous scientific instruments, from self-driving labs to agent-supervised synchrotron runs, generate provenance at machine speed: dense in decisions, multi-pass, and shared between people and agents, yet largely invisible to the humans who must later audit it. A scientist returning to an unattended overnight run asks plain questions: did the alignment converge, what was in flight when the beam dropped, who or what decided to resume, and can I prove what was recorded? Post-hoc provenance graphs flatten the temporal and causal texture those questions depend on. Event-sourced systems of record answer them internally, by folding an immutable event stream to reconstruct any past state; we turn that fold into an interaction. We contribute the conjunction of deterministic event-stream replay, provenance visualization, a content-addressed fidelity check run at the cursor, and a set of audit tasks for autonomous runs, realized as a *replay scrubber*: a draggable time cursor that replays a run to any recorded instant. It shows who or what drove each transition, verdict-colored convergence bands, open-interval markers for unfinished steps, and a badge that verifies the replayed recipe expansion matches the content hash recorded at run start. We ground the prototype in an integration-tested, agent-supervised scenario run on an APS 2-BM micro-CT configuration, against the real event-sourcing stack: the system conducts a rotation-axis centering alignment, a supervisor agent holds the run on beam loss and auto-resumes it, and the scrubber localizes the interrupted projection by folding to the beam-loss instant. The run is agent-supervised rather than agent-closed: the agent recovers a conducted run, it does not select the science. We discuss how event-sourced records reshape provenance visualization for autonomous science.

Index Terms: Provenance visualization, event sourcing, interactive replay, autonomous science, agent-supervised experiments, content-addressed integrity, audit, human-agent accountability.

1 INTRODUCTION

A beamline increasingly runs itself overnight. An operator queues a sample, hands the instrument to an automated routine, and goes home; software aligns the sample, starts acquiring, and a supervisor agent watches the beam and holds or resumes the run as conditions change. When the scientist returns in the morning, they still have to answer for what happened while no one was watching. The first questions are plain: did the alignment converge, what was in flight when the beam dropped at 2am, who or what decided to resume, and can I prove the data was taken under the conditions I think it was?

These are provenance questions, but the provenance of an autonomous run is unlike the human-authored record most tools were built for. It arrives at machine speed, it is dense in decisions, it is multi-pass, and control passes back and forth between people and agents. Post-hoc provenance graphs and flat logs flatten exactly the temporal and causal texture these questions depend on. A graph drawn after the run can tell you that steps happened and how they connect, but not what the run looked like at two in the morning, and it gives the auditor nowhere to stand inside the run and look around. When the beam drops, the log simply stops, leaving the auditor to guess what was in flight and whether it took effect; a static record offers no way to show that it has not been altered after the fact; and nothing says who was driving at each moment. Read this way, auditing an autonomous run is a visual analytics problem for autonomous science: it is about oversight, replay, audit, and human-agent accountability, the question of how a person reconstructs and answers for what an instrument and its agents did unattended.

We observe that one class of system already answers such questions internally. An event-sourced system of record never stores current state directly; it reconstructs any past state on demand by folding an immutable, ordered stream of events. Folding is the everyday operation it sounds like: replay the log from the beginning, apply each event in turn, and the state re-forms, the way re-adding a ledger's entries recovers its running balance. Our central idea is to expose that fold as an interaction. The result is a *replay scrubber*: a draggable time cursor that an auditor moves through a recorded run, with the system folding the event stream to the chosen instant and rendering the state as it was then, who or what caused it, and whether the reconstruction is faithful. Because the view is driven by the same deterministic fold used by the backend read path, the displayed state is reconstructed from the recorded event prefix rather than re-created by a separate visualization-specific replay; and a content-addressed check of the recorded recipe expansion lets the scrubber verify, rather than merely assert, that the expansion shown at the cursor matches the hash recorded at run start (under the system's append-only and deterministic-fold assumptions, not adversarial tamper resistance, and covering the recipe expansion rather than the full dynamic run state).

We ground the design in an integration-tested, agent-supervised scenario run of our event-sourced system of record, CORA, on an APS 2-BM micro-CT configuration. The scenario exercises the real event-sourcing stack: it conducts a rotation-axis centering alignment that converges, the science scan begins and a projection is in flight when the beam drops, and a deterministic supervisor agent holds the run and auto-resumes it when the beam returns. The evidence chain is explicit: a passing integration scenario runs against the real Kernel and Postgres event store; its recorded activities, iteration verdicts, lifecycle events, and supervisor decisions are mirrored by a standalone generator into the run data the figures render, with only the per-event timestamps staggered synthetically for a readable axis. We are deliberate about scope. The agent supervises and recovers a *conducted* run, recording each hold and resume as a decision; the fully agent-closed perceive-decide-act loop, in which a model selects the science itself, is out of scope, and the prototype is a research visualization over exported run data rather than a deployed interface. We treat that boundary as a discussion opening rather than a limitation to apologize for: because a closed-loop optimizer would record its next-action choices through the same decision substrate, the supervised case is the near end of a progression, the open question of what oversight, replay, and accountabil-

*Corresponding author. e-mail: dgursoy@anl.gov

ity should look like as automation moves from agent-supervised toward agent-closed.

The novelty is not a timeline or time travel as such, but their conjunction in this setting: deterministic event-stream replay, provenance visualization, a content-addressed fidelity check run at the cursor, and a task abstraction for auditing autonomous runs.

This paper makes four contributions:

- **Replay-native provenance (insight).** We reframe the fold-on-read mechanism of an event-sourced system of record as an interaction primitive for provenance: the fold the backend uses to reconstruct state becomes a control the auditor drags through recorded time, rather than a static post-hoc graph to read.
- **A task abstraction for auditing autonomous runs.** We characterize four recurring audit tasks, recover an interrupted step, reproduce a decision, confirm convergence, and confirm integrity, and map each to an event-log query and a visual encoding, separating the audit need from the encoding we chose for it.
- **The replay scrubber (technique).** We design a single timeline that unifies a run-lifecycle lane showing who or what drove each transition, verdict-colored convergence bands, open-interval markers for unfinished steps, a draggable fold-to-version cursor, and a content-addressed fidelity badge.
- **A grounded case study.** We demonstrate the four tasks on an integration-tested, agent-supervised scenario run on an APS 2-BM micro-CT configuration, exercised against the real event-sourcing stack, including the beam-loss instant at which the scrubber surfaces the agent that held the run and the interrupted projection it left open.

2 The Auditor's Tasks

We start from the questions, not the interface: before designing anything, we ask what a scientist actually wants to know on returning to a run that ran itself, and let those needs drive the encodings. We derive four recurring tasks, which we label T1 through T4, from what a beamline scientist needs when returning to a conducted run, and make each concrete as a query over the event log paired with a visual encoding (Table 1). The tasks are not hypothetical: each is a question we ask of the agent-supervised run in Section 3, and each is answerable only because the record is an ordered, immutable event stream rather than a final-state snapshot.

T1: recover the interrupted state. *What was in flight when the beam dropped?* The system records intent before effect, so an unfinished step appears in the log as an activity that was opened but never closed. The query finds the activity with no matching outcome; the encoding draws it as an open interval, a visible gap rather than a silent truncation. The same record also tells an interrupted step apart from one never reached: only an interruption leaves an opened-but-unclosed marker.

T2: reproduce a decision. *Replay the run to the instant a choice was made, and show the state then.* The query folds the event stream up to the chosen position; the encoding is the draggable cursor together with the reconstructed state read out beside it. This recovers the state the choice was made against, not merely its timestamp, which is what makes a decision reviewable after the fact.

T3: confirm convergence. *Did this loop converge, and on which pass?* The query reads the per-iteration verdict carried by each iteration-boundary event; the encoding shades each iteration's band by that verdict, so the search's progress is legible at a glance. For an unattended search, reading the converged verdict is the difference between trusting the result and re-running the alignment to check it.

Table 1: Four auditor tasks, each mapped to an event-log query and a visual encoding. The mapping is the spine of the design.

Task	Event-log query	Visual encoding
T1: Recover interrupted state	Activity opened with no closing outcome	Open interval (a visible gap)
T2: Reproduce a decision	Fold the stream to position t	Draggable cursor + folded state
T3: Confirm convergence	Per-iteration verdict on boundary events	Verdict-colored iteration bands
T4: Confirm integrity	Recompute and compare the recorded hash	Inline fidelity badge

T4: confirm the record is intact. *Can I trust that nothing was altered?* The query recomputes the content-addressed hash recorded with the run and compares it at the cursor; the encoding is an inline fidelity badge that turns "trust me" into "the view checked it."

These four tasks order the rest of the paper: the scrubber (Section 3) demonstrates all four, with T2 and T3 as its core interactions; the interruption-recovery walkthrough (Section 4) develops T1; and the immutable substrate (Section 5) is what lets T4's badge verify rather than assert.

3 The Replay Scrubber

Before the mechanics, the experience. The scrubber turns a finished run into something a scientist can play back: drag a handle along time and the run re-forms at that instant, the way a video scrubber jumps to a frame.

The scrubber presents one recorded run on a single horizontal time axis (Figure 1), a stack of lanes that each answer a different question. The activity swim-lanes carry one track per kind of step the run performs, a position setpoint, an acquisition, a check, and the dataset write to disk, with every activity drawn at its recorded time within its lane. Above them a run-lifecycle lane records the autonomous arc, who or what caused each transition, colored by actor, so an auditor sees at a glance where control passed between the operator and the supervisor agent; and a beam-permit lane shows the safety envelope the supervisor gates on, satisfied except across the hold. Color-coded bars across the top name the run's phases (alignment, scan, save). The iterative structure of the conducted alignment is shown as one shaded band per pass, each tinted by that pass's recorded verdict and grouping the pass's setpoint, acquire, and check. An activity that was opened but never closed, the record of intent before effect, is drawn as an open interval rather than a filled mark, so an unfinished step reads as a visible gap.

The interaction is a single draggable cursor. Setting the cursor to a position t folds the event stream up to t and renders the reconstructed state of the run beside the timeline, so dragging the cursor replays the run to any recorded instant (T2). The fold is the system's own read path, not a separate reconstruction built for the figure. Where most timelines let a user read *when* something happened, the scrubber lets them recover *what the run knew* at that moment, and *who or what* was driving it.

A small badge travels with the cursor (Figure 2). At each position it recomputes the content-addressed expansion hashes recorded with the run and compares them against the reconstructed state, reporting whether the two match (T4). The badge turns the integrity property of the underlying store, that the record is append-only and the expansion is content-addressed, into something a reader can see at the exact instant they are inspecting, rather than a claim they must take on faith.

We walk through an unattended, agent-supervised run on APS 2-BM. An operator starts the run and leaves; the system conducts the

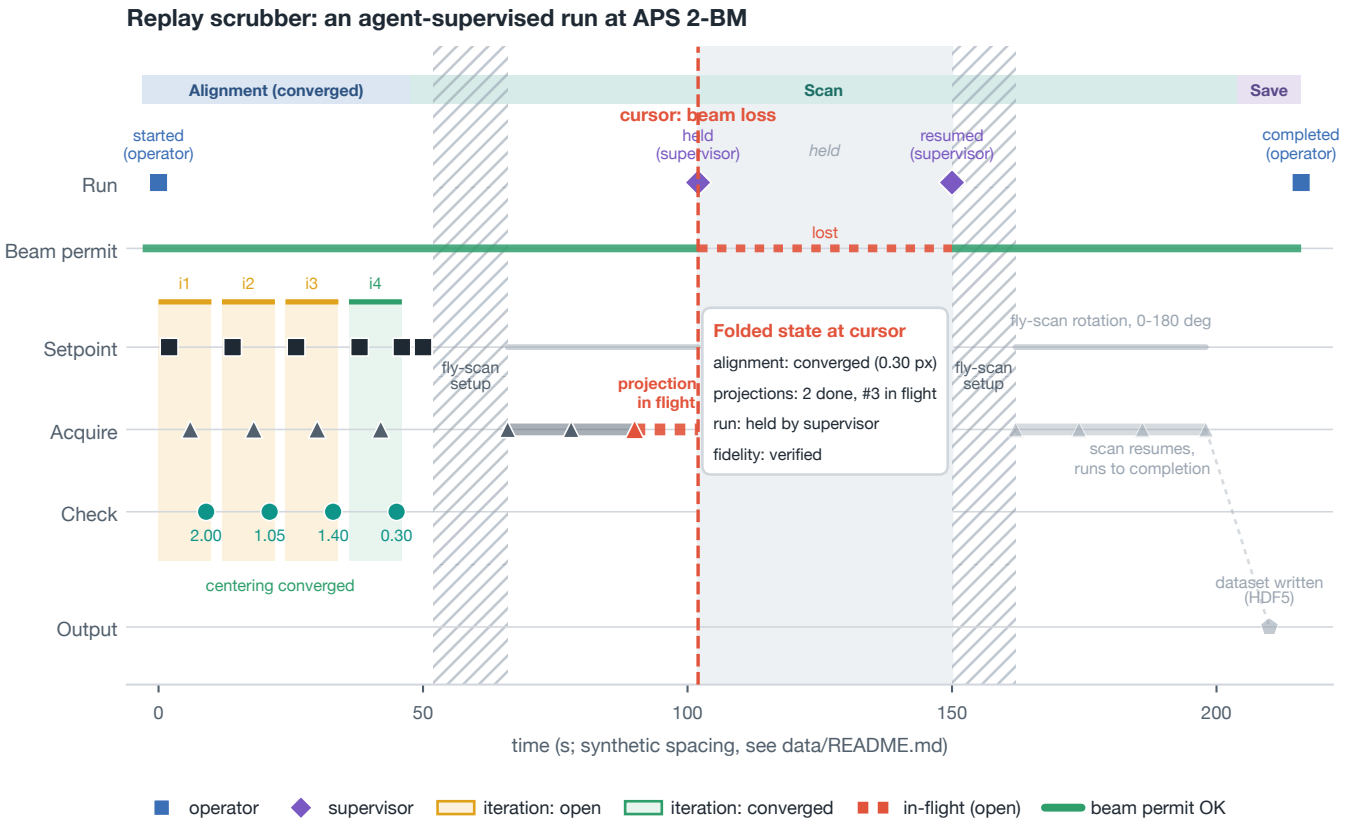

Figure 1: The replay scrubber over one agent-supervised run at APS 2-BM. Color-coded bars across the top name the run's phases (alignment, scan, save). A run-lifecycle lane records who drove each transition: the operator started and the run completes, while the supervisor agent held the run when the beam dropped and resumed it when the beam returned (the gray band is the hold; the two hatched bands are the fly-scan setup, the rotary taxi to constant velocity and trigger arm, needed before the first frame and again after the hold). A beam-permit lane shows the safety envelope the supervisor gates on: satisfied except across the hold, where it reads lost. Below, the setpoint, acquire, and check swim-lanes, with one verdict-tinted band per pass of the conducted rotation-axis centering search (amber open, green converged) grouping that pass's three activities. The fold-to-version cursor is parked at the beam-loss instant: folding the event stream to there reconstructs the state shown in the read-out, including the third projection as an open interval (its in-flight marker is recorded but its outcome is not yet). Past the cursor, after the run resumes, the scan runs to completion and the dataset is written to disk. Time spacing is synthetic.

pre-scan rotation-axis centering alignment, a four-iteration peak-bracket search over the sample stage `SampleTop_X` that minimizes the center-of-rotation residual. The residual falls from 2.00 to 1.05 px, rises to 1.40 px at 0.080 mm (bracketing the minimum in $[0.040, 0.080]$ mm), and the fourth pass bisects to 0.060 mm at 0.30 px and converges; read by color, the iteration bands show the verdict sequence open, open, open, converged (T3). The science scan begins; two projections complete and the third is in flight when the beam drops. The supervisor agent holds the run; the beam returns and the supervisor resumes it, each transition recorded as the agent's decision in the lifecycle lane. With the cursor at the beam-loss instant (Figure 1), the fold shows the alignment already converged, two projections done and the third still open, and the run held by the agent, and the fidelity badge confirms that the reconstructed recipe expansion is faithful to the hash recorded at run start (T4).

Read as a morning audit, the same run answers the auditor's questions in the order they arise. Dragging the cursor into the gap in the held band localizes the third projection as an open interval, exactly the exposure in flight when the beam dropped, rather than guessing it (T1, developed in Section 4). Dragging to the resume transition opens its decision, the supervisor resumed because the start-safety envelope was satisfied again, and the folded state shows what the agent saw at that instant (T2). In a few minutes, and without re-running anything, the auditor has recovered what the run did overnight, what was interrupted, and why the agent acted.

Formalization. The mechanism behind that walkthrough is a single fold. Write $e_{1:t}$ for the event prefix up to cursor t, and let $\mathrm{evolve}(S, e)$ be the transition function that applies one event e to a state S and returns the next state, starting from the empty state S_0. The reconstructed state is the deterministic left fold $S_t = \mathrm{fold}(\mathrm{evolve}, S_0, e_{1:t})$, which applies evolve to each event in turn; it is the same transition the backend read path uses, so the cursor reaches every recorded instant exactly. At run start the system records two content-addressed hashes of the recipe expansion, $h^{\star}_{\mathrm{steps}}$ and $h^{\star}_{\mathrm{bind}}$. The badge reads verified at t when both recompute to their recorded values: $\mathrm{hash}(\mathrm{steps}(S_t)) = h^{\star}_{\mathrm{steps}}$ and $\mathrm{hash}(\mathrm{bind}(S_t)) = h^{\star}_{\mathrm{bind}}$. Since evolve only folds recorded events, S_t reproduces the record rather than re-running the alignment.

Design rationale. Each encoding answers an audit question rather than decorating the timeline. We give the run lifecycle its own lane, colored by actor, instead of annotating events in place, so that *who or what was driving* is legible across the whole run and accountability is separated from activity. We draw convergence as verdict-colored iteration bands rather than a metric curve: the auditor is asking whether and on which pass the search decided it had converged, which is the verdict, not the residual value. We place the fidelity badge inline at the cursor rather than in a separate trust panel, so trust is judged at the instant being inspected. We render an unfinished step as an open interval rather than truncating the

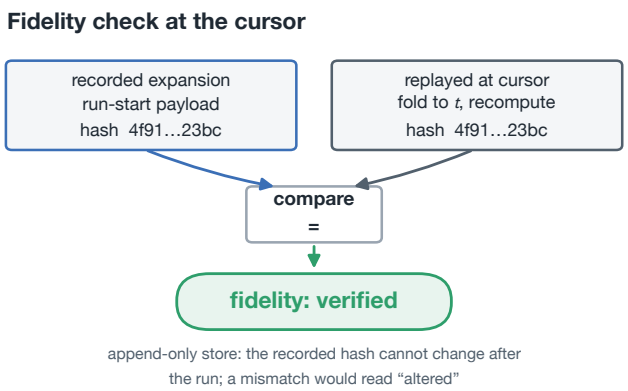

Figure 2: The fidelity check the badge runs at the cursor. The content-addressed hashes recorded at run start, over the recipe expansion (the resolved steps and the parameter bindings), are recomputed from the state reconstructed by folding to the cursor and compared; a match reads verified, a mismatch would read altered. The digest shown is an illustrative content hash of the run data.

timeline, so a missing outcome is a positive mark that is seen rather than inferred. And we keep one linear time axis with an explicit held band rather than collapsing the hold, because the gap itself is the evidence that the run waited, on the same scale as everything else.

Auditing autonomous agents. The lifecycle lane, and the decision records behind it, make the scrubber an instrument for agent accountability, which is the harder half of the human-agent handoff. Each transition the supervisor drives is recorded with the actor that caused it and a decision the auditor can open: not only that the run was held and resumed, but who or what decided, when, and on what basis. Scrubbing to a transition reconstructs the state the agent saw at that instant, so “why did it resume here” is answered by replay rather than by trusting a log line. Accountability is not only about the acquisition: the same fold reconstructs every axis the system records as events, so the beam-permit lane shown here, and the clearance, training, and authorization boundaries the system records elsewhere, are recovered at the cursor alongside the science, and an auditor scrubs the safety envelope that gated a decision as readily as the data it produced. The questions an auditor asks do not change as autonomy deepens; only how many of the recorded decisions were the agent’s rather than a person’s. The same lane that here distinguishes operator from supervisor would, with more agents in the loop, separate their contributions and surface where they contended or handed off, the multi-agent orchestration view that prior provenance tools do not provide.

4 Interruption Recovery

The most operationally pressing question on an unattended run is also the one conventional records answer worst: when the beam drops overnight, what was in flight, and did it take effect? The system records intent before effect, so a side-effecting acquisition is written as an in-flight marker before the exposure and an outcome after. When the supervisor holds the run mid-acquisition, the marker is on the record but the outcome is not yet, so folding the stream to the beam-loss instant reconstructs a state in which exactly that step is open, and the scrubber draws it as an open interval (Figure 3, top). The auditor sees precisely which step was mid-flight, not a timeline that simply ends.

What makes this replay-native rather than a stored flag is that the same step reads as completed once the cursor moves past the resume: after the beam returns and the supervisor resumes, the outcome is appended, and folding to the end reconstructs the projection as closed and the run as completed (Figure 3, bottom). The

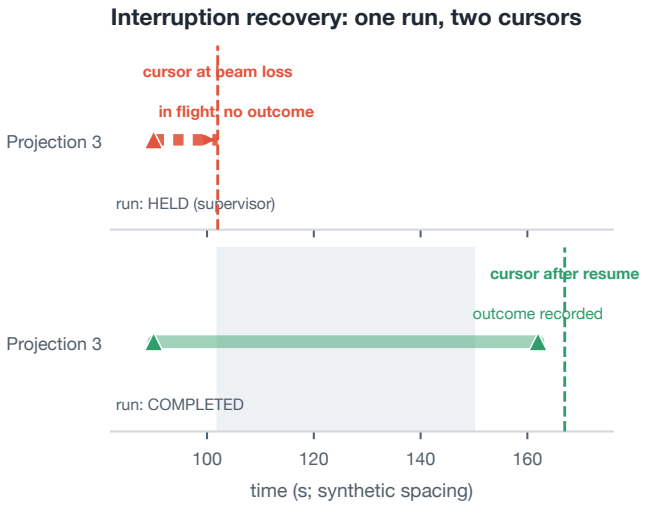

Figure 3: The same recorded run folded to two cursor positions. At the beam-loss instant (top) the third projection is an open interval, an in-flight marker with no outcome, and the run is held by the supervisor; folded to the end (bottom) the outcome is recorded and the run is completed. The open interval is a function of where the cursor is, not a permanent dangling record: it is the replay-native answer to what was in flight when the beam dropped (T1).

open interval is not a permanent record of failure; it is what the fold yields at an instant when the outcome had not yet happened. This is the case post-hoc provenance graphs handle least gracefully: a graph assembled after the fact has no node for an action that has not reported a result, whereas the intent-before-effect record, read by replay, makes the missing result legible at exactly the instants where it is missing.

5 Substrate

A run is not stored as current state: it is an append-only, totally ordered stream of events, and any state the scrubber shows is produced by folding the prefix up to a chosen position. The scrubber needs nothing from the backend beyond what an event-sourced system of record already keeps. We summarize the substrate here, and stress that it is the enabling condition, not the contribution (Figure 4).

Three records carry everything the four tasks need. Each event’s envelope records who or what caused it and its position in the stream, which gives the cursor a well-defined instant to fold to (T2) and gives an unfinished activity a position to mark as an open interval (T1). The run-start payload records the recipe the run expanded from, together with content-addressed hashes of that expansion; recomputing and comparing those hashes is exactly the fidelity badge (T4). The per-iteration boundary events carry the verdict that colors each iteration band (T3). The swim-lane activities, the setpoint, acquire, check, and output rows, are themselves events appended as the system acts, with the intent recorded before the effect (the open-interval mechanism of Section 4). Concretely, the example run is a four-event run-lifecycle stream (started, held, resumed, completed) alongside a twelve-event procedure stream with around two dozen activity rows; folding a prefix returns the reconstructed run and procedure state (status, the current iteration and its verdict, and whether each activity is open or closed), and the recorded hashes cover the recipe expansion, the resolved steps and parameter bindings, written at run start.

The visual claim, that the timeline you are scrubbing is faithful to an unaltered record under stated assumptions, is what the paper is about; we enumerate those assumptions below rather than re-prove them here.

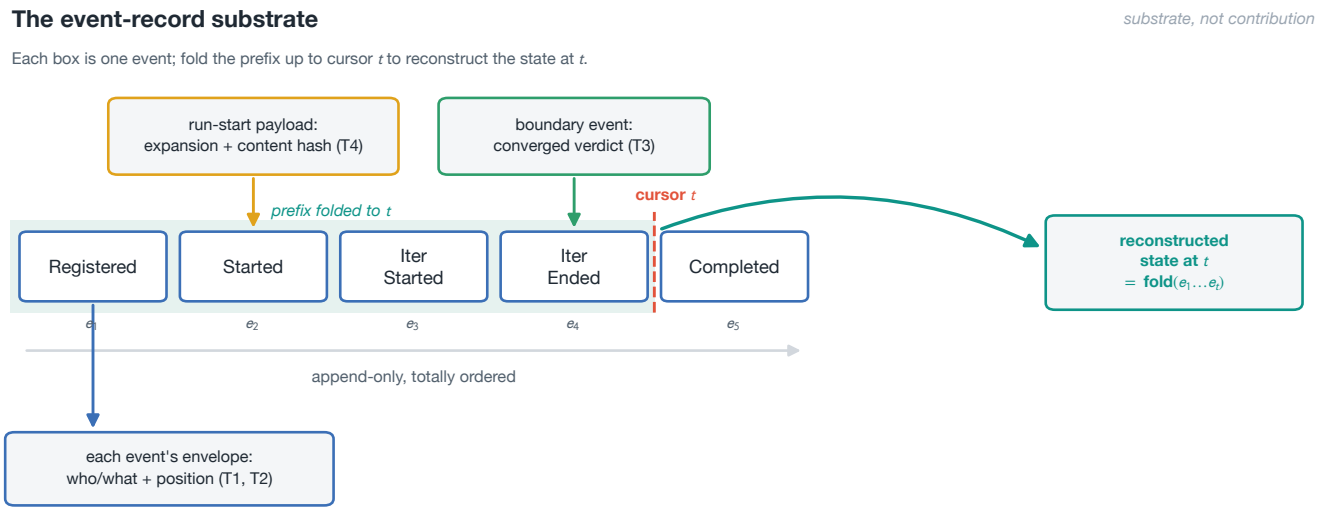

Figure 4: The event-record substrate behind the scrubber. A run is an append-only, totally ordered event stream; folding the prefix up to the cursor reconstructs the state shown. Three records carry what the tasks need: each event's envelope (who or what caused it, and its position), the run-start payload (the recipe expansion and its content hash, feeding the fidelity badge), and the iteration-boundary events (the converged verdict).

Table 2: The four records the scrubber reads from the event log. The scrubber needs nothing beyond these, which is why the technique transfers to any append-only, deterministically-foldable event store.

Record	Key fields
Event envelope	stream id, position, actor (human or agent), timestamp
Run-start payload	plan reference, parameter bindings, content-addressed expansion hashes
Iteration boundary	iteration index, converged verdict, reason
Activity entry	step kind, payload, result (e.g. in-flight or ok), sampled-at

The records the scrubber reads are summarized in Table 2.

Assumptions. The fidelity claim holds under four conditions, stated here so it is not read more strongly than intended: the fold is deterministic (the same event prefix always reconstructs the same state); storage is append-only, so the recorded side is not editable through the normal application path; the event writer is trusted; and the hash inputs are stable. It is an integrity check under these conditions, not adversarial tamper resistance: an attacker with direct write access to the store is out of scope, and authenticity against forgery would need a separate trust anchor. A faithfully recorded run therefore always reads verified; the badge reads altered only when the reconstruction has drifted from the run it describes, through a change to how state is rebuilt, a schema migration, or a corrupted store, which makes it a continuous regression check on the read model rather than a decorative light.

6 Related Work

We locate our contribution in the gap none of the neighboring work fills: a deterministic fold over an immutable event record, exposed as a scrub interaction, carrying a content-addressed fidelity check and provenance semantics for autonomous loops. The comparison table carries the per-system contrasts (Table 3); here we walk the four lines of related work and where each stops short.

Provenance visualization, replay, and time-travel. Retrospective provenance has long been distinguished from the prospective recipe that specifies it [21], and tools that visualize it, such as VisTrails [10] and Loops [9], let an analyst navigate and diff recorded versions. Verdant [15, 14] is the closest prior interaction: it pairs recorded notebook history with a draggable timeline, but its replay re-executes recorded versions rather than folding an immutable stream, so it cannot certify the match, and it targets a local notebook. QIIME 2 Provenance Replay [13] regenerates code to re-run offline rather than offering an interactive view. Omniscient debuggers such as Pernosco and rr [24], the Whyline [16], and reducer-based state containers share our record-once, replay-later posture, but hold deterministic replay as an internal guarantee never rendered to the user. These systems replay computation; we replay recorded run state with provenance semantics and a visible, checkable fidelity affordance.

Tamper-evident provenance and reproducibility badges. The canonical integrity structures, Crosby and Wallach's history tree [7], Certificate Transparency's Merkle logs [18], and content-addressed identifiers such as SWHID [8], verify membership or recompute-and-compare as machine-checked properties with no user interface. The closest human-facing analog is C2PA Content Credentials [6], recently carried to machine-learning pipelines [28] as a content-addressed hash-versus-reference check at a programmatic service layer over discrete artifacts, with no visualization and no replayed state. Reproducibility-record tools, CPR [23], Whole Tale [5], workflow-run provenance crates [20], and artifact badges [2], emit static reports or certify a run through human audit within a tolerance. None reconstructs per-instant state or renders an exact, content-addressed match. We render a content-addressed verification verdict to a human, recomputed as the user scrubs, over reconstructed run state.

Conformance checking. Process mining supplies the canonical verdict of whether a run conformed, through token-replay fitness [26], alignment diagnostics [3, 4], and online variants on a live stream [30]. These verdicts are encoded as static overlays on a process model or per-move coloring along a completed trace, never along an iteration axis with interval structure, and post-hoc alignments have no representation for an interrupted step. Recent surveys conclude that the visual encoding of conformance results remains an open problem [25, 11]. We contribute exactly such an encoding: verdict-colored convergence bands along the iteration axis, with open intervals for in-flight steps.

Table 3: Where the replay scrubber sits relative to the closest prior work. • = present, ○ = absent or not surfaced to the user. *Replay* names the reconstruction mechanism; *fidelity* is an integrity check, over the replayed recipe expansion, recomputed at the cursor and shown to a human; *run-state provenance* is per-iteration verdict and in-flight semantics along a run; *who / handoff* is a per-transition actor (human or agent) record; *scrub* is an interactive time cursor. C2PA/Atlas read ○ on fidelity because their check is over static artifacts, not replayed run state; the distinction is the per-cursor reconstruction, not the hash. No prior system combines all five columns.

System	Replay	Fidelity	Run-state prov.	Who / handoff	Scrub
Verdant [15]	re-execute	○	○	○	•
Pernosco / rr [24]	record-fold	○	○	○	•
QIIME 2 Replay [13]	code-gen, offline	○	○	○	○
Conformance checking [26, 4]	token-replay	○	○	○	○
C2PA / Atlas [6, 28]	none	○	○	○	○
Workflow-Run RO-Crate [20]	none	○	○	○	○
Replay scrubber (ours)	deterministic fold	•	•	•	•

Visual analytics for autonomous experimentation. Self-driving-lab and autonomous-beamline systems keep operations logs and present live dashboards: VISION [22] for backend monitoring, and optimizer and campaign tools such as Olympus [12], the autonomous SAXS/WAXS framework at ALS and PETRA III [17], Tsuchinoko [19], ChemOS 2.0 [27], A-Lab [29], and Bluesky's live callbacks [1] render score fields, convergence lines, and live plots. These are forward-streaming views of a run in progress; none offers a scrubbable, post-hoc audit of the recorded run, per-iteration convergence verdicts, or a fidelity check. That gap is what the replay scrubber addresses.

Open questions. We see the scrubber less as a finished answer than as a way to open three questions. First, what should a fidelity check cover as autonomy deepens: we verify the recipe expansion, but a closed-loop run invites debate over which dynamic state (decisions, model inputs, next-action choices) should be made content-addressable, and at what cost to legibility. Second, how should oversight scale to many agents: the lifecycle lane that here separates one operator from one supervisor would need encodings for contention, handoff, and blame across several agents at once. Third, what is the right division of labor between post-hoc audit and live steering, since the same folded timeline could host a now-edge for monitoring a run in progress. We offer the scrubber as a concrete substrate against which these can be argued out.

7 Limitations and Future Work

Our evaluation is a case-based demonstration: a structured walkthrough of the four auditor tasks on one recorded run. It shows feasibility and explanatory power, but it is not an empirical validation, and it does not establish that the scrubber improves audit speed or accuracy. We report a single run, on a single instrument, and we have not yet run a user or expert study; whether the scrubber speeds real audits, and which encodings practitioners prefer, are open empirical questions. The interface is a research prototype over exported run data rather than a deployed beamline tool, so we make no claims about throughput, scale, or integration cost.

The figures are not mock-ups. They derive from one passing integration scenario, exercised against a real Kernel and Postgres event store, and the run data, its generator, and the per-figure render scripts are released with the paper. What is real is what that scenario produced and we mirrored into the figure data: the activities, the four iteration verdicts (open, open, open, converged), the four run-lifecycle events, and the supervisor's recorded hold and resume decisions. What is synthetic is only the per-event timestamps, staggered for a readable axis because the scenario records one logical instant; the overnight wall-clock spread is illustrative, stated in prose rather than measured. The generator depends on nothing but the standard library and needs no database, so a reader can regenerate the figure data directly.

Two boundaries matter for how the contribution should be read, both about what the run and the badge are not. The run is *conducted* and *agent-supervised*, not agent-closed: the system orchestrates the alignment and records every step, and a supervisor agent holds and resumes the run on beam conditions, but no model is in the loop choosing what to measure or how to steer it. That supervisor is a deterministic, rule-based agent making recorded decisions, so the figures show a genuine human-agent handoff rather than a model selecting experiments, and we are careful not to claim the latter. The fidelity badge likewise attests *faithfulness*, that the replayed state matches the recorded expansion, by recomputing content-addressed hashes against an append-only store; it is an integrity check, not adversarial tamper resistance, under the conditions stated in Section 5.

The fold-to-version helper that drives the cursor was written for this visualization on top of the system's existing fold-on-read path. The fold is scoped to one run's stream, not the whole store, so the work is bounded by a single run rather than the entire event history; but because that path keeps no snapshots, the cost of reconstructing a distant instant still grows with the length of that stream. For the short run shown here it is imperceptible, and for long campaigns the standard remedies apply directly, periodic snapshots to fold from and incremental re-folding rather than replaying from the start each time; we name these rather than build them. Three further extensions are natural: a live now-edge that subscribes to the event tail so the same timeline can monitor a run in progress, run-to-run diffing so two campaigns share one axis, and a deployed read interface, whose first read-only slice is already on the roadmap. The closest extension is an agent-closed steered loop, an optimizer that decides the next action rather than a supervisor that recovers a conducted one; because such an optimizer would record its choices through the same decision substrate, the run-lifecycle lane and the decision records would carry its next-action choices and be audited exactly as the supervisor's holds and resumes are here. We expect the design to generalize in the same way beyond this run, to other conducted procedures and other facilities, since nothing in the scrubber is specific to this case beyond a small vocabulary it reads from the log: the requirement on a host system is modest, an append-only, deterministically-foldable event log with a content-addressed expansion and a recorded actor on each event.

Three threats temper these claims. The construct question is whether T1 through T4 capture what auditing a conducted run actually needs; we derived them from practice rather than from observing auditors at work, so the set may be incomplete, missing for example causal blame across steps or finer parameter provenance, or may favor an encoding scientists would not. There is also an internal threat, since the walkthrough is authored by us, who already know where each verdict, open interval, and gap lies; an independent auditor reading the same figures cold might not find them as readily, and we ran no such study. Finally, generality is the external threat: our argument for it is structural, not demonstrated beyond the single beamline, run, and rule-based supervisor reported here. These open questions set the agenda for the user study and multi-beamline deployment that should follow.

References

[1] D. Allan, T. Caswell, S. Campbell, and M. Rakitin. Bluesky's ahead: A multi-facility collaboration for an a la carte software project for data acquisition and management. *Synchrotron Radiation News*, 32(3):19–22, 2019. doi: 10.1080/08940886.2019.1608121 6

[2] Association for Computing Machinery. Artifact review and badging, version 1.1. https://www.acm.org/publications/policies/artifact-review-and-badging-current, 2020. 5

[3] A. Berti and W. M. P. van der Aalst. Reviving token-based replay: Increasing speed while improving diagnostics. In *Algorithms & Theories for the Analysis of Event Data (ATAED)*, vol. 2371 of *CEUR Workshop Proceedings*, 2019. 5

[4] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst. Process mining for python (pm4py): Bridging the gap between process- and data science. arXiv:1905.06169, 2019. 5, 6

[5] A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski, V. Stodden, I. J. Taylor, M. J. Turk, and K. Turner. Computing environments for reproducibility: Capturing the "whole tale". *Future Generation Computer Systems*, 94:854–867, 2019. doi: 10.1016/j.future.2017.12.029 5

[6] Coalition for Content Provenance and Authenticity. C2PA technical specification. https://spec.c2pa.org/, 2024. Content Credentials Verify: https://verify.contentauthenticity.org/. 5, 6

[7] S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, 2009. 5

[8] R. Di Cosmo, M. Gruenpeter, and S. Zacchiroli. Identifiers for digital objects: the case of software source code preservation. In *International Conference on Digital Preservation (iPRES)*, 2018. doi: 10.17605/OSF.IO/KDE56 5

[9] K. Eckelt, K. Gadhave, A. Lex, and M. Streit. Loops: Leveraging provenance and visualization to support exploratory data analysis in notebooks. *IEEE Transactions on Visualization and Computer Graphics (VIS)*, 2025. doi: 10.1109/TVCG.2024.3456186 5

[10] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *Provenance and Annotation of Data (IPAW)*, vol. 4145 of *LNCS*, 2006. doi: 10.1007/11890850_2 5

[11] M.-C. Häge and J.-R. Rehse. Conformance checking visualizations: development, evaluation, and demonstration of a taxonomy. *Process Science*, 2025. doi: 10.1007/s44311-025-00025-5 5

[12] F. Häse, M. Aldeghi, R. J. Hickman, L. M. Roch, M. Christensen, E. Liles, J. E. Hein, and A. Aspuru-Guzik. Olympus: a benchmarking framework for noisy optimization and experiment planning. *Machine Learning: Science and Technology*, 2(3):035021, 2021. doi: 10.1088/2632-2153/abedc8 6

[13] C. R. Keefe, M. R. Dillon, E. Gehret, C. Herman, M. Jewell, C. V. Wood, E. Bolyen, and J. G. Caporaso. Facilitating bioinformatics reproducibility with qiime 2 provenance replay. *PLOS Computational Biology*, 19(11):e1011676, 2023. doi: 10.1371/journal.pcbi.1011676 5, 6

[14] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2019. doi: 10.1145/3290605.3300322 5

[15] M. B. Kery and B. A. Myers. Interactions for untangling messy history in a computational notebook. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018. doi: 10.1109/VLHCC.2018.8506576 5, 6

[16] A. J. Ko and B. A. Myers. Finding causes of program output with the Java Whyline. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2009. doi: 10.1145/1518701.1518942 5

[17] W. Koepp, B. Sochor, D. McReynolds, T. Chavez, M. Noack, R. V. Sriramoju, A. H. Coffey, Y. Wang, E. Henn, A. K. Sambale, E. Euchler, D. English, F. Schlünzen, E. Schaible, C. Zhu, S. Koyiloth Vayalil, S. V. Roth, and A. Hexemer. Toward unified autonomous scattering experiments: A cross-facility case study at ALS and PETRA III. *Photon Science*, 2026. doi: 10.1021/photonsci.5c00044 6

[18] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013. doi: 10.17487/RFC6962 5

[19] Lawrence Berkeley National Laboratory, CAMERA. Tsuchinoko: an adaptive-experiment gui. https://tsuchinoko.readthedocs.io/, 2024. OSTI code-135210. 6

[20] S. Leo, M. R. Crusoe, L. Rodríguez-Navas, R. Sirvent, A. Kanitz, P. De Geest, R. Wittner, L. Pireddu, D. Garijo, J. M. Fernández, et al. Recording provenance of workflow runs with ro-crate. *PLOS ONE*, 19(9):e0309210, 2024. doi: 10.1371/journal.pone.0309210 5, 6

[21] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. Prospective and retrospective provenance collection in scientific workflow environments. In *IEEE International Conference on Services Computing (SCC)*, 2010. doi: 10.1109/SCC.2010.18 5

[22] S. Mathur, N. van der Vleuten, K. G. Yager, and E. H. R. Tsai. VISION: a modular AI assistant for natural human-instrument interaction at scientific user facilities. *Machine Learning: Science and Technology*, 6, 2025. doi: 10.1088/2632-2153/add9e4 6

[23] T. M. McPhillips, T. Thelen, C. Willis, K. Kowalik, M. B. Jones, and B. Ludäscher. CPR: A comprehensible provenance record for verification workflows in Whole Tale. In *Provenance and Annotation of Data and Processes (IPAW)*, vol. 12839 of *LNCS*, pp. 263–269, 2021. doi: 10.1007/978-3-030-80960-7_23 5

[24] R. O'Callahan, C. Jones, N. Froyd, K. Huey, A. Noll, and N. Partush. Engineering record and replay for deployability. In *USENIX Annual Technical Conference (ATC)*, 2017. 5, 6

[25] J.-R. Rehse, L. Pufahl, M. Grohs, and L.-M. Klein. Process mining meets visual analytics: The case of conformance checking. In *Hawaii International Conference on System Sciences (HICSS-56)*, 2023. arXiv:2209.09712. 5

[26] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008. doi: 10.1016/j.is.2007.07.001 5, 6

[27] M. Sim, M. Ghazi Vakili, F. Strieth-Kalthoff, H. Hao, R. J. Hickman, S. Miret, S. Pablo-García, and A. Aspuru-Guzik. ChemOS 2.0: An orchestration architecture for chemical self-driving laboratories. *Matter*, 7, 2024. doi: 10.1016/j.matt.2024.04.022 6

[28] M. Spoczynski, M. S. Melara, and S. Szyller. Atlas: A framework for ML lifecycle provenance and transparency. arXiv:2502.19567, 2025. 5, 6

[29] N. J. Szymanski, B. Rendy, Y. Fei, R. E. Kumar, T. He, D. Milsted, M. J. McDermott, M. Gallant, E. D. Cubuk, A. Merchant, H. Kim, A. Jain, C. J. Bartel, K. Persson, Y. Zeng, and G. Ceder. An autonomous laboratory for the accelerated synthesis of inorganic materials. *Nature*, 624:86–91, 2023. doi: 10.1038/s41586-023-06734-w 6

[30] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8:269–284, 2019. doi: 10.1007/s41060-017-0078-6 5