Protean and Galen: Provenance-Preserving Bounded Scientific Cognition for Peptide Discovery

Protean Labs

Draft revised 2026-05-22

Abstract

AI-assisted scientific workflows increasingly combine retrieval, language models, persistent memory, and task orchestration, but many systems do not make evidence lineage, replayability, memory mutation, or external-action boundaries explicit. We describe Protean and Galen, a local-first runtime for peptide discovery workflows that treats provenance and bounded orchestration as systems invariants. Protean records immutable cycle manifests, artifact hashes, redacted public provenance graphs, deterministic object and edge identifiers, and lifecycle-aware lineage surfaces. Galen coordinates bounded cognition over evidence indexing, hybrid retrieval, optional local reranking, claim assessment, contradiction signals, scientific memory consolidation, research planning, telemetry, and HMAC-gated remediation. The implementation grounds these mechanisms in peptide workflows including candidate generation, motif and cleavage analysis, sequence-space mapping, developability heuristics, claim QA, and computational experiment planning. We report the implemented architecture and identify current limits: workflow DAG enforcement is feature-flagged and not the default scheduler, contradiction graphs are implemented and consumed when present but are not yet materialized as a default loop stage, wetlab coordination is review-only scaffolding with no provider execution, and biological validation is not claimed. The contribution is a reproducible systems pattern for continuous bounded scientific cognition with provenance-preserving replayable lineage, not autonomous laboratory discovery.

1 Introduction

AI-assisted scientific workflows now combine literature retrieval, language-model reasoning, candidate generation, scoring, persistent memory, and operational automation. This composition creates a systems problem. A scientific runtime must preserve which evidence influenced each output, make memory updates inspectable, distinguish deterministic control from model inference, and prevent computational plans from becoming unreviewed external actions. In computational biology these requirements are acute: evidence is heterogeneous, negative results matter, candidate sequences may be sensitive, and wetlab actions require review.

Protean and Galen address this problem as runtime infrastructure rather than as an autonomous discovery claim. Protean is the provenance layer. It records immutable cycle snapshots, artifact manifests, redacted public graphs, deterministic scientific object identifiers, and lifecycle-aware lineage. Galen is the bounded cognition and orchestration layer. It runs local-first evidence retrieval, reranking, claim QA, hypothesis generation, scientific memory consolidation, research planning, telemetry, state transitions, and approval-gated remediation. The system is designed to continue synthesizing evidence while preserving replayability and operator control.

The current implementation targets peptide discovery workflows. It generates and ranks peptide candidates, maps sequence-space neighborhoods, identifies motif families, flags cleavage and failure

motifs, evaluates candidate explanations against local evidence, proposes bounded hypotheses, and emits computational experiment plans. It does not perform autonomous wetlab execution, does not publish raw sequences, does not self-modify runtime code, and does not claim validated receptor binding, PK/PD prediction, efficacy, or clinical utility.

The system's design point is continuous bounded scientific cognition with provenance-preserving replayable lineage. Continuous cognition means that the runtime can ingest evidence, consolidate memory, and propose new local artifacts across cycles. Bounded cognition means that policy limits recursion, learning passes, reranking passes, write roots, provider escalation, publication, and wetlab actions. Replayable lineage means that outputs can be traced through manifests, hashes, graph nodes, and edge identifiers back to the evidence and cycle artifacts that produced them.

This paper makes the following contributions:

1. A provenance-first runtime architecture for AI-assisted peptide discovery, implemented with immutable cycle manifests, redacted public graphs, deterministic identifiers, and explicit public-private boundaries.

2. A bounded cognition pipeline that combines evidence retrieval, optional local reranking, claim QA, contradiction signals, hypothesis generation, scientific memory consolidation, and research planning without allowing memory to mutate validators, scoring rules, prompts, or code.

3. A review-gated orchestration model with finite runtime states, HMAC-signed approval receipts, idempotent approval consumption, allowlisted remediation executors, and hard refusals for wetlab submission, provider packet send, payment dispatch, raw sequence disclosure, and self-modification.

4. A peptide-focused workflow that grounds the architecture in candidate generation, sequence-space analysis, motif and cleavage reasoning, developability heuristics, and computational experiment planning.

5. An implementation status map that distinguishes implemented, partial, feature-flagged, planned, and intentionally disabled subsystems.

2 Related Work

Scientific workflow systems. Galaxy, Taverna, Kepler, Snakemake, Nextflow, and the Common Workflow Language provide executable workflow graphs, dependency management, and reproducible computational pipelines [5, 11, 3, 17, 15, 1]. Protean and Galen share the emphasis on explicit artifacts and repeatable execution but operate at a different layer. Their workflow outputs include evidence bundles, ranked candidates, claim assessments, hypotheses, memory snapshots, approval records, and provenance graphs. The current runtime does not yet use its declared DAG as the default scheduler; workflow guard enforcement is implemented but feature-flagged. This is a narrower claim than mature workflow engines make, but it exposes an integration path between scientific cognition artifacts and conventional workflow provenance.

Provenance and research object frameworks. W3C PROV, RO-Crate, DataLad, MLflow, FAIR data principles, and related systems describe data derivation, research object packaging, experiment tracking, and reproducible computational environments [16, 19, 6, 23, 22]. Protean is complementary. It is not a general provenance standard; it is a domain runtime that records scientific cognition artifacts and public-private graph boundaries. It adds deterministic candidate

commitments, redacted graph reconstruction, lifecycle and disclosure states, and review-linked operational records for peptide workflows.

Retrieval-augmented scientific systems. Retrieval-augmented generation and dense retrieval systems condition generation on retrieved evidence [13, 9]. Scientific QA and claim-verification datasets such as PubMedQA and SciFact motivate evidence-grounded biomedical and scientific answering [7, 21]. Galen uses local hybrid retrieval, optional local cross-encoder reranking, claim QA, and source-trust enrichment, but treats retrieval traces as runtime artifacts rather than transient prompt context. Contradiction signals can influence hypotheses and planning when graph artifacts are present, but the system does not implement a complete truth-maintenance system.

LLM scientific agents. Recent agent systems use language models for literature review, code execution, hypothesis generation, and experiment planning. Protean and Galen deliberately avoid the stronger claim that an agent autonomously conducts science. The runtime is local-first, review-gated, and bounded by explicit policy. Memory can inform retrieval and planning but cannot rewrite scoring rules or validators. External actions require review or are refused. This places the system closer to auditable scientific infrastructure than to open-ended autonomous research agents.

Autonomous laboratories and self-driving experimentation. Self-driving laboratories and robotic science systems automate experiment selection, execution, and feedback in controlled settings [10, 2]. Protean and Galen do not currently operate a laboratory, submit assays, accept quotes, dispatch payment, or ingest wetlab results through a completed executor. The wetlab layer is a review-only interface design. The relevant comparison is therefore not throughput or closed-loop experimental optimization, but action boundary management: computational plans remain local artifacts until an operator reviews and coordinates external work.

Computational biology and peptide infrastructure. Protein language models, sequence embeddings, curated databases, and structural predictors support modern protein and peptide workflows [18, 14, 8, 20, 4]. The current peptide workflow uses local sequence features, motif analysis, failure records, and optional sequence embeddings. It does not claim validated receptor targeting, PK/PD simulation, toxicology, immunogenicity, or efficacy prediction. Its contribution is an auditable runtime around these signals rather than a new biological model.

3 System Architecture

3.1 Runtime boundary

The implementation is organized around a local cycle runner, a bounded cognition cluster, deterministic provenance writers, and governance modules. The cycle owner is `infra/loop_runner.py`. A full cycle performs local health checks, optional bounded online ingestion, entity extraction, evidence indexing, offline ingestion, normalization, candidate generation, feature extraction, pattern learning, scoring, optional candidate explanations, claim QA, bounded learning feedback, optional single reranking, research cognition, provenance graph construction, collection updates, and cycle snapshotting. A cognition-only cycle runs the research cognition cluster without ingestion, candidate generation, scoring, or learning.

Galen modules under `agents/orchestration/`, `agents/governance/`, and `agents/telemetry/` handle state, policy, approvals, remediation, and telemetry. Protean modules under

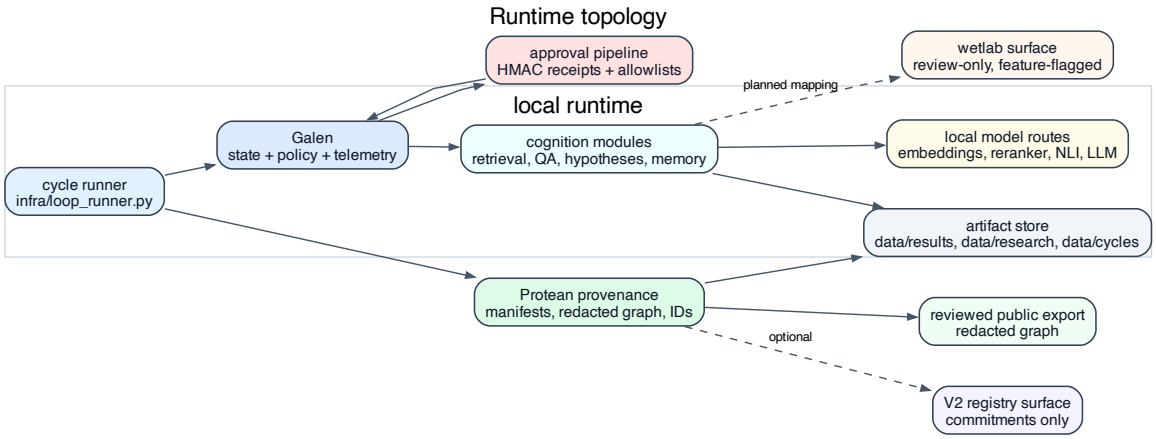

Figure 1: Runtime topology. The scheduler, Galen policy/state/telemetry, cognition modules, local model routes, artifact store, Protean provenance layer, approval pipeline, public export, and feature-flagged wetlab boundary are separated explicitly.

`pipelines/provenance_graph.py`, `pipelines/onchain_ids.py`, runtime artifact writers, and contract definitions handle lineage, deterministic identifiers, public-private boundaries, and registry-compatible commitments.

3.2 Implementation status

Table 1 summarizes the implemented boundary. The status column is part of the claim, not ancillary documentation.

3.3 Full runtime cycle example

A representative full cycle uses a timestamped identifier such as `20260522T181045Z`. The runner writes or updates working artifacts during execution and then copies selected outputs into `data/cycles/20260522T181045Z/`. The manifest `data/cycles/20260522T181045Z/run_manifest.json` records each artifact, producer, byte count, row count, and SHA-256 digest. Typical artifacts include:

- `data/results/ranked_candidates.csv`, produced by `pipelines.score`;
- `data/results/claim_qa_latest.jsonl`, produced by `pipelines.claim_qa`;
- `data/research/hypotheses.jsonl`, produced by `pipelines.hypothesis_engine`;
- experiment index artifact, produced by `pipelines.experiment_planner`;
- public provenance graph artifact, produced by `pipelines.provenance_graph`.

The cycle then updates pointer manifests such as `data/runtime/current_cycle_manifest.json` and `data/results/latest_run_manifest.json`. If a non-critical cognition stage fails, the cycle records a warning and may enter a degraded state; it does not silently promote missing cognition artifacts to valid outputs.

Subsystem	Status	Boundary
Cycle runner and snapshots	Implemented. Full and cognition-only cycles write immutable cycle artifacts.	Cycle success does not imply biological validity.
Runtime manifests	Implemented. SHA-256 hashes, byte counts, row counts, producers, and validation pointers.	Validates artifact integrity, not external source truth.
Public provenance graph	Implemented as redacted graph schema `protean.scientific-object-graph.v1`.	V1 public schema; not all V2 object types emitted.
V2 object and edge IDs	Implemented in Python utilities and Solidity contracts; deployed on Base Sepolia.	Mainnet blocked by governance, audit, and operational review.
Evidence index	Implemented with SQLite, embeddings, lexical overlap, and sequence-query blocking.	Quality depends on local corpus and model availability.
Reranker	Implemented as optional local cross-encoder path.	Falls back to lexical or index order when unavailable.
Claim QA	Implemented with local NLI when available and heuristic fallback.	Evidence-relative assessment, not proof of truth.
Contradiction graph	Implemented and tested; consumed when present.	Not default loop materialization; no transitive closure.
Scientific memory	Implemented as bounded artifact consolidation.	Cannot mutate scoring, validators, prompts, or code.
Workflow DAG guard	Implemented and feature-flagged.	Enforcement disabled by default and not integrated as scheduler.
Capability ontology	Implemented as read-only records and inspectors.	Not enforced by experiment planner or wetlab mapping.
Wetlab coordination	Data classes and review-only draft packet agent implemented.	No provider execution, assay submission, quote acceptance, or payment.
Approval pipeline	Implemented with signed receipts, allowlists, idempotency, and telemetry.	Only a small remediation executor set is active.
External providers	Local-first policy implemented.	Remote provider escalation is not production-enabled.

Table 1: Implementation status and claim boundaries.

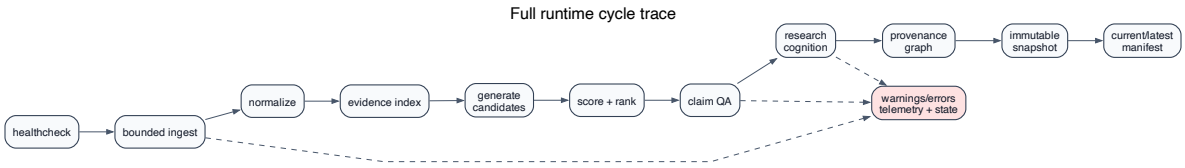

Figure 2: Full runtime cycle trace. The trace runs from health checking through bounded ingest, evidence indexing, candidate scoring, claim QA, research cognition, provenance graph construction, immutable snapshotting, and pointer manifest update.

4 Provenance and Replayable Lineage

4.1 Manifest model

Each full cycle ends with an immutable cycle directory. The writer refuses to overwrite a nonempty cycle directory. Artifacts are copied atomically into the cycle directory and recorded in `run_manifest.json`. A schematic entry is:

```
{
  "cycle_id": "20260522T181045Z",
  "artifacts": {
    "claim_qa": {
      "path": "data/results/claim_qa_latest.jsonl",
      "sha256": "91...",
      "bytes": 61204,
```

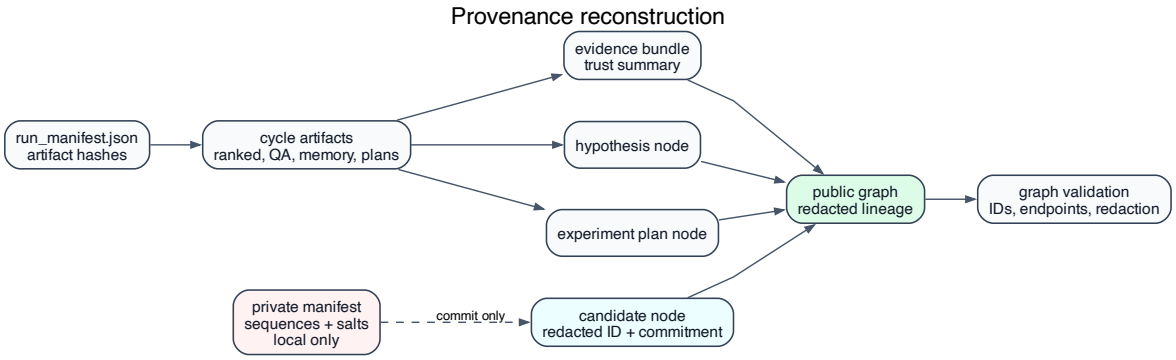

Figure 3: Provenance reconstruction flow. Private artifacts and manifests produce redacted public graph nodes and edges; validation checks artifact hashes, graph identifiers, edge endpoints, and public redaction invariants.

```
      "row_count": 50,
      "producer": "pipelines.claim_qa"
    }
  }
}
```

Replay validation rehashes each artifact and compares it with the manifest. This validates local artifact integrity and detects missing or mutated files. It does not certify that an upstream database has not changed or that a peptide claim is biologically true.

4.2 Public-private graph boundary

The provenance graph separates private work product from public lineage. Private manifests retain raw sequences, salts, and detailed candidate records. Public graph nodes expose redacted candidate identifiers, commitments, content hashes, lineage hashes, and reviewed summaries. Candidate commitments use a private salt so short peptide sequences are not exposed to direct public hash enumeration. The graph validator checks duplicate identifiers, missing endpoints, redaction violations, and graph digest consistency.

4.3 Deterministic identifiers

Protean uses deterministic identifiers for scientific objects and lineage edges. The current public graph builder emits a V1 graph schema. The V2 identifier utilities and contracts expand the ontology to additional object classes including approval receipts, provider packets, review records, capability artifacts, scientific memory snapshots, public releases, assay batches, quote reviews, payment intents, and external signals. V2 object and edge identifiers use explicit domain separators:

```
PROTEAN_SCIENTIFIC_OBJECT_V2
PROTEAN_LINEAGE_EDGE_V2
```

The V2 registry surface stores commitments and pointers rather than raw sequences or payment information. A Base Sepolia deployment exists, but mainnet use remains blocked by operational review, audit, and governance hardening. Public-chain anchoring is treated as a commitment mechanism, not as scientific validation.

4.4 Replay validation flow

A replay check consists of:

1. Select a cycle manifest, for example `data/cycles/20260522T181045Z/run_manifest.json`.
2. Recompute SHA-256 for each listed artifact.
3. Compare byte counts and row counts where available.
4. Rebuild or verify `data/provenance/graph/provenance_graph_latest.json`.
5. Validate graph invariants: no duplicate node IDs, no missing edge endpoints, no raw-sequence leakage in public nodes, and stable graph digest.
6. Compare pointer manifests against the selected cycle if verifying the latest runtime state.
7. Inspect telemetry records for operational warnings, model fallbacks, approval decisions, or remediation actions that occurred during the cycle.

This procedure gives a reproducible artifact trail for scientific review. It does not require trusting a language-model explanation as the record of what happened.

5 Cognition Pipeline

5.1 Evidence retrieval and reranking

The evidence index is a local SQLite store over evidence records, literature mentions, and failure records. Retrieval combines text embeddings with lexical overlap and blocks peptide-like sequence queries. The shared retrieval helper applies a bounded candidate ceiling and optional local cross-encoder reranking. Rerank annotations include method, score, and rank. When the reranker is disabled or unavailable, the system uses lexical or index fallback and records the fallback method.

Retrieval example. For a query such as `protease stability proline shielding cleavage`, the retriever reads `data/cache/evidence_index.sqlite`, returns bounded candidate snippets, and optionally reranks them. A downstream claim QA record may include fields such as `retrieval_method`, `rerank_method`, `rerank_rank`, `source_trust_score`, and `freshness_score`. If the query is a raw peptide-like sequence, retrieval refuses the query and routes sequence analysis to sequence-space modules instead.

5.2 Claim QA and contradiction routing

Claim QA compares generated claims against retrieved evidence using local NLI when available and heuristic fallback otherwise. It treats prohibited or unsupported categories, such as therapeutic efficacy, patent certainty, and unmeasured pharmacokinetic improvement, conservatively. Its output is evidence-relative.

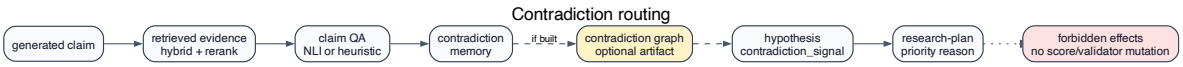

Figure 4: Contradiction-routing flow. Claim QA emits evidence-relative contradiction records. When a contradiction graph artifact exists, consumers can attach hypothesis signals and research-plan priority reasons without mutating scores or validators.

Contradiction-routing example. Suppose a candidate explanation claims that candidate `cand_20260522_014` "improves pharmacokinetics." The editorial and claim QA layers route this as an unsupported developability claim unless local evidence exists. Claim QA writes an entry to `data/results/claim_qa_latest.jsonl`. If the contradiction graph builder is run, it can aggregate this record into `data/research/memory/contradiction_graph.json`. The hypothesis engine can then attach a `contradiction_signal` to related hypotheses, and the research planner can add a priority reason such as `contradiction_graph_signal`. The signal changes prioritization context; it does not delete evidence, rewrite scoring, or mutate validators.

5.3 Hypotheses, experiments, and memory

The hypothesis engine emits bounded hypothesis classes: motif, cleavage, shielding, novelty, failure-correlation, and structural hypotheses. These hypotheses are generated from candidate groups, motif families, cleavage heuristics, failure records, evidence snippets, and optional contradiction signals. Hypothesis status remains shallow; most records are proposed or marked as insufficient. Richer lifecycle transitions are future work.

The experiment planner emits computational plans such as mutation sweeps, motif perturbation studies, contrastive sets, local exploration branches, and ablation challenges. These plans are review artifacts. They do not become wetlab orders.

Scientific memory consolidates lineage memory, retrieval history, motif memory, pattern memory, contradiction memory, exploration memory, experiment memory, memory events, and a memory manifest. Memory can inform retrieval and research planning. Governance policy prevents it from mutating scoring rules, validators, learning rules, prompt templates, or code.

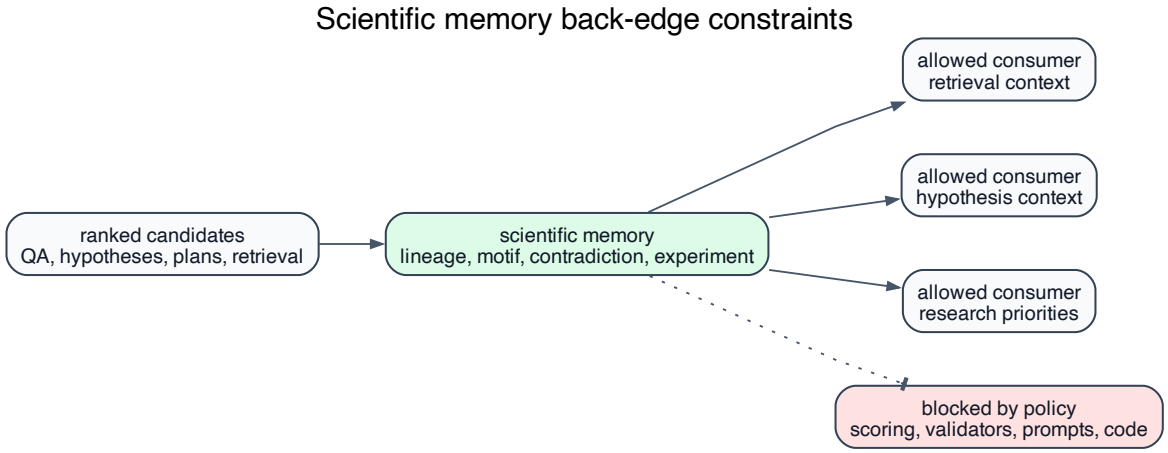

Figure 5: Scientific memory and constrained back-edge. Memory artifacts can inform retrieval, hypothesis context, and research planning, but policy blocks score mutation, validator mutation, prompt rewriting, code rewriting, and raw sequence publication.

6 Peptide Discovery Workflow

6.1 Implemented peptide operations

The implemented peptide workflow supports candidate generation, deterministic validation, feature extraction, ranking, sequence-space mapping, claim QA, hypothesis generation, memory consolidation, and computational experiment planning. Candidate generation combines stable local motifs, rotated seeds, balancing residues, cleavage-site protection heuristics, novelty checks, failure similarity checks, and validators. Ranking computes heuristic developability proxies including solubility, permeability, synthesis risk, novelty, stable similarity, failure similarity, and failure motif penalties.

Sequence-space analysis computes pairwise distances, clusters, motif families, neighborhoods, redundancy groups, novelty gradients, length bands, charge bands, composition labels, and unexplored regions. Motif-family artifacts are written under `data/research/sequence_space/`. These outputs support review and planning; they do not establish biological activity.

Peptide evidence-routing example. A candidate family with shared motif `PxxP`, a failure motif warning, and moderate novelty can produce the following trace:

1. the ranked-candidate artifact records scores and failure motif warnings.
2. the motif-family artifact records the candidate family and neighborhood.
3. `data/results/claim_qa_latest.jsonl` records whether generated claims are supported, unsupported, or contradicted.
4. `data/research/hypotheses.jsonl` records a cleavage or shielding hypothesis if evidence and candidate context support one.
5. `data/research/experiments/experiments_latest.json` records a motif perturbation or mutation-sweep plan.

6. the public provenance graph links evidence bundle, redacted candidate node, hypothesis, and experiment plan.

6.2 Non-implemented biological claims

The current runtime does not implement validated receptor-specific binding prediction, receptor docking, pathway-level mechanism inference, full PK/PD simulation, toxicology, immunogenicity assessment, or efficacy prediction. Receptor and PK/PD terms may appear as extracted entities, evidence-routing targets, or measurement prompts, but the system should not phrase them as conclusions unless external evidence supports them.

7 Runtime Safety and Boundedness

7.1 Governance policy

Galen's governance policy bounds maximum workflow steps, recursion depth, learning passes per cycle, reranking passes per cycle, retrieval top-k, experiment candidates, online ingestion results, and provider escalations. It restricts writable roots to runtime data directories and forbids source-code mutation. It hard-refuses rewrite-runtime-code, scoring mutation, learning-rule mutation, recursive prompt rewriting, raw sequence publication, review bypass, wetlab submission, quote acceptance, provider packet send, payment dispatch, and self-modifying architecture.

7.2 Approval and refusal behavior

Approval receipts are HMAC-SHA256 signatures over canonical JSON [12]. Outside development mode, missing or weak signing keys fail closed. The approval consumer verifies signature, expiry, idempotency, and action allowlist membership before dispatch. The bare remediation path without an approval identifier is refused.

Approval/refusal example. An approved `rebuild_evidence_index` action can be represented in `data/agents/approval_queue.jsonl`, signed into a private receipt under `data/private/security/approval_log.jsonl`, consumed once, and recorded in `data/telemetry/remediation.jsonl`. By contrast, an action such as `wetlab.assay_submit`, `wetlab.payment_dispatch`, `publish_raw_sequences`, or `code.rewrite` is refused even if it appears in an approval-like envelope.

7.3 Wetlab boundary

Wetlab interfaces define assay requests, assay batches, provider packets, quote reviews, payment intents, assay results, and review records. A review gate rejects auto-submit, auto-accept-quote, and skip-draft modes. The wetlab agent prepares redacted draft provider packets for review. It does not contact providers or submit assays. Result-ingest and review-close action names exist for future allowlisted handling, but real executors and payload schemas are not complete.

7.4 Runtime state machine

The state machine includes healthy, warning, degraded, blocked, awaiting review, awaiting publication, awaiting assay, assay running, assay complete, and failed states. Deterministic state derivation currently emits the health and review subset based on manifest validation, review-gated artifacts,

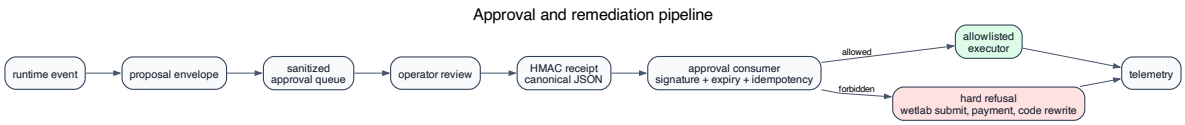

Figure 6: Approval and remediation pipeline. Runtime events become proposal envelopes and sanitized approval records. Signed receipts are checked for signature, expiry, idempotency, and allowlist membership before execution or refusal.

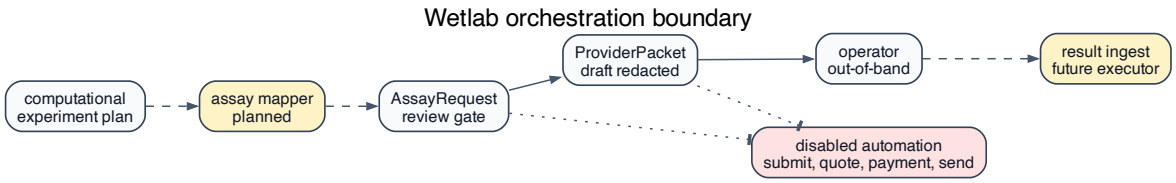

Figure 7: Wetlab orchestration boundary. Current wetlab code supports review-gated data structures and draft provider packets only; provider execution, quote acceptance, payment dispatch, and assay submission are absent.

and runtime health. Explicit writers exist for blocked, awaiting publication, and assay states. Assay writers are not invoked by the default loop.

8 Operational Characteristics

The default loop can run once, continuously, or in cognition-only mode. Continuous mode sleeps between cycles and gates online ingestion by cadence. The loop records warnings and errors from stages and can degrade rather than terminate on non-critical failures. Model routes are local-first, and modules use deterministic or heuristic fallback where implemented. Telemetry records operational events under `data/telemetry/`; scientific artifacts remain separate from operational logs.

The workflow DAG is declared as twelve steps, including evidence retrieval, hypothesis formation, experiment planning, candidate generation, validation, ranking, explanations, claim QA, paper generation, provenance preparation, collection update, and lineage update. The guard can validate dependencies, allowed writes, and expected outputs when enabled. Since the default loop does not yet enforce the DAG per stage, DAG semantics are partial infrastructure.

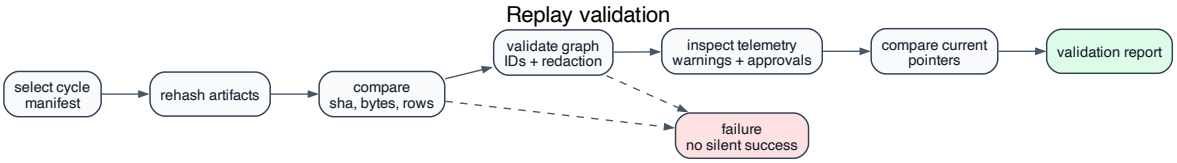

Figure 8: Replay validation flow. A reviewer selects a cycle manifest, rehashes artifacts, compares metadata, validates graph invariants, inspects telemetry, and checks current/latest pointers.

9 Evaluation Plan and Available Checks

This paper evaluates infrastructure properties, not biological discovery. Where quantitative results are not yet collected, we specify the experimental method and mark the measurement as pending.

9.1 Available repository checks

The following checks are implemented and were run during this revision:

- `python3 agents/orchestration/galen.py -verify`: verifies agent registry, Galen identity, local scheduler ownership, local-first routes, provider policy, no self-modification, runtime health model, anomaly model, state machine, Telegram boundedness, and workflow DAG declaration.
- `python3 pipelines/provenance_graph.py -verify`: validates public graph structure, edge endpoints, redaction invariants, and graph schema.
- `python3 agents/orchestration/workflow_guard.py -status`: reports the DAG enforcement flag and confirms that enforcement is disabled by default.
- Targeted unit tests for contradiction graph behavior, cognition back-edge activation, workflow guard behavior, and remediation executors.

9.2 Manifest replay determinism

Method: select a set of historical cycle directories, re-run manifest validation, and report the fraction of artifacts whose SHA-256, byte count, and row count match the manifest. Also report validation wall-clock time and missing-artifact categories. Current status: implemented validation path; aggregate benchmark pending.

9.3 Graph integrity

Method: rebuild or verify `data/provenance/graph/provenance_graph_latest.json`; measure node count, edge count, duplicate ID count, missing endpoint count, redaction violations, graph digest stability, and validation time. Current observed check: the latest graph validated with schema `protean.scientific-object-graph.v1`; aggregate longitudinal graph evaluation pending.

9.4 Retrieval and reranking

Method: construct a fixed query set covering protease stability, cleavage motifs, solubility, synthesis risk, negative evidence, and assay context. Compare lexical-only, embedding-only, hybrid, and hybrid plus local cross-encoder reranking. Report latency distribution, top-k overlap, reranker

Measurement	Observed value	Source
Cycle manifests discovered	163	from data/cycles/*/run_manifest.json
Latest cycle ID	loop-9-1779329503	data/cycles/loop-9-1779329503/run_manifest.json
Latest cycle artifact count	35	manifest artifacts
Manifest validation	2.769 ms	0 problems
Graph validation	2.76 ms	33 nodes / 32 edges
Graph schema	protean.scientific-object-graph.v1	latest public graph
Telemetry records	267	all data/telemetry/*.jsonl

Table 2: Repository-derived runtime and replay measurements collected during the submission build. These are infrastructure checks, not biological benchmarks.

fallback frequency, and expert-labeled relevance when labels are available. Current status: retrieval and reranking implemented; curated benchmark labels pending.

9.5 Contradiction routing

Method: seed claim QA with supported, unsupported, and contradicted statements drawn from local evidence and failure records. Measure claim-label stability, graph aggregation correctness, hypothesis `contradiction_signal` attachment, and research-plan priority changes with and without the contradiction graph artifact. Current status: graph builder and consumers implemented; default-loop graph materialization pending.

9.6 Memory back-edge constraints

Method: run cycles with controlled memory artifacts and verify that retrieval, hypotheses, or research-plan priorities can change while scoring weights, validators, prompt templates, and source files remain unchanged. Compare file hashes for protected code and configuration. Current status: governance rules and tests cover portions of this boundary; complete longitudinal evaluation pending.

9.7 Approval and refusal correctness

Method: submit signed and unsigned approval receipts for allowlisted and forbidden actions. Report acceptance, refusal, idempotency behavior, telemetry emission, and executor side effects. Include malformed signatures, expired receipts, duplicate receipts, and missing signing keys outside development mode. Current status: approval consumer and remediation executors implemented for a limited action set; full adversarial matrix pending.

9.8 Telemetry and cycle overhead

Method: measure event volume, manifest size, graph size, memory artifact size, cycle duration, graph validation time, and manifest validation time across cycles. Partition runtime cost by ingestion, indexing, generation, scoring, claim QA, cognition, provenance graphing, and snapshot writing. Current status: telemetry exists; aggregate cost reporting pending.

9.9 Failure recovery

Method: inject stale locks, missing optional local models, malformed evidence rows, graph validation errors, missing artifacts, and disabled reranker models. Verify degraded states, warning propagation, refusal of unsafe remediation, and correctness of approved recovery actions. Current status: selected remediation executors tested; broad fault-injection evaluation pending.

10 Limitations

The system has several material limitations.

Contradiction graphing is incomplete. Claim QA, contradiction memory, and a deterministic graph builder exist, and consumers can use graph signals when present. The graph is not yet materialized as a default loop stage, lacks transitive closure, and does not drive a mature hypothesis lifecycle.

Ontology integration is partial. V2 object and edge identifiers and contracts exist, but the public graph remains V1-style and several V2 object types have no default writer. The capability ontology is readable but not enforced by experiment planning or wetlab mapping.

Workflow orchestration is partial. The workflow DAG and guard are implemented, but enforcement is feature-flagged off and not the default scheduler. The current loop is procedural.

Wetlab execution is absent. The repository defines review-gated wetlab data structures and draft provider packets, but there is no provider adapter, assay submission, quote acceptance, payment dispatch, or completed result-ingest executor.

The runtime is not self-modifying. This is a safety property and a limitation. Memory cannot change validators, scoring rules, prompts, learning rules, or source code.

Biological validation is not provided. Candidate scores are heuristic prioritization signals. The system does not validate efficacy, receptor binding, PK/PD, toxicology, immunogenicity, or clinical utility.

Retrieval is corpus-limited. Local evidence coverage, embedding availability, source parsing quality, and reranker availability affect output quality. Source-trust scoring is auditable but not a substitute for expert curation.

Public-chain anchoring, where used, is a commitment mechanism. It does not validate scientific truth and is not required for local replay.

11 Future Work

Near-term work should integrate implemented scaffolding into default-loop behavior without expanding external autonomy. Contradiction graph materialization should become a first-class loop artifact. Hypotheses should gain lifecycle transitions such as supported, contradicted, refuted, revised, and superseded. Capability records should be enforced in experiment planning and wetlab request mapping.

Wetlab development should proceed through a simulated dry run before provider integration: experiment plan to assay request, draft provider packet, operator review, simulated result import, provenance graph update, and memory consolidation. Real provider adapters should remain disabled until the dry run is reproducible and reviewed.

Evidence ingestion should expand to curated assay and structure sources after terms and provenance review. Semantic reconciliation should improve linking between peptide names, sequences, assay contexts, protein targets, mechanisms, and source identifiers. Evaluation should move from

correctness checks to measured latency, replay stability, retrieval quality, contradiction precision, refusal behavior, and operational overhead.

12 Conclusion

Protean and Galen implement a local-first architecture for continuous bounded scientific cognition in peptide discovery workflows. Protean records immutable manifests, redacted graphs, deterministic identifiers, and lifecycle-aware lineage. Galen coordinates bounded retrieval, reranking, claim QA, contradiction signals, hypothesis generation, scientific memory, planning, telemetry, state transitions, and signed remediation.

The core result is an auditable runtime pattern: scientific cognition can operate continuously while preserving replayable evidence lineage and review gates. The implementation remains intentionally constrained. It does not automate wetlab execution, self-modify, or claim biological validation. Its value is the reproducible infrastructure boundary it provides for AI-assisted scientific work.

A Reproducibility Appendix

A reproducibility bundle should contain:

- `data/cycles/<cycle_id>/run_manifest.json`;
- all artifacts referenced by the manifest;
- `data/provenance/graph/provenance_graph_latest.json`;
- `data/runtime/current_cycle_manifest.json`;
- relevant `data/telemetry/*.jsonl` files;
- redacted public export files if the bundle is public;
- private salt and raw sequence files only in private review bundles.

Recommended validation commands:

```
python3 agents/orchestration/galen.py --verify
python3 pipelines/provenance_graph.py --verify
python3 agents/orchestration/workflow_guard.py --status
```

For a private replay, rehash every manifest artifact and compare graph digest, node count, edge count, and redaction checks. For a public replay, reconstruct only from reviewed redacted artifacts and verify that private sequence fields are absent.

B Runtime Trace Appendix

Recommended trace table columns:

- cycle identifier;
- stage name;

Cycle	Stage	Outputs	Warnings/errors	State	Replay status
loop-9-1779329503	generation	3 artifacts; e.g. candidates-csv	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	scoring	2 artifacts; e.g. ranked-candidates	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	explanations	2 artifacts; e.g. candidate-explanations	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	sequence-space	6 artifacts; e.g. motif-families	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	research-retrieval	1 artifacts; e.g. research-retrieval	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	scientific-memory	8 artifacts; e.g. contradiction-memory	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	experiment-planner	2 artifacts; e.g. experiment-index	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	research-planner	2 artifacts; e.g. research-plan	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	source-provenance	3 artifacts; e.g. evidence-provenance	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	data-expansion	1 artifacts; e.g. data-expansion-plan	3 warnings / 0 errors	awaiting-review	pass
loop-9-1779329503	bounded-learning	3 artifacts; e.g. feedback-signals	3 warnings / 0 errors	awaiting-review	pass

Table 3: Runtime trace reconstructed from the latest cycle manifest and telemetry state record. Output names are manifest artifact keys, not generated prose.

- producer module;
- input artifacts;
- output artifacts;
- warnings or errors;
- state transition;
- telemetry event identifiers;
- manifest digest after snapshot.

The trace should be generated from runtime manifests and telemetry rather than from model-generated prose.

C Artifact Release Structure

Recommended arXiv companion artifact layout:

```
artifact/
  README.md
  environment/
    python_version.txt
    package_lock_or_freeze.txt
  manifests/
    run_manifest.example.json
    current_cycle_manifest.example.json
  provenance/
    provenance_graph_redacted.example.json
    graph_validation_report.example.json
  telemetry/
    telemetry_schema.md
    redacted_events.example.jsonl
  schemas/
    evidence_record.schema.json
    claim_qa.schema.json
```

```
    hypothesis.schema.json
    memory_manifest.schema.json
    approval_receipt.schema.json
  scripts/
    verify_manifest.py
    verify_graph.py
  figures/
    fig01_runtime_topology.pdf
    fig02_runtime_cycle_trace.pdf
    fig03_provenance_reconstruction.pdf
    fig04_contradiction_flow.pdf
    fig05_memory_constraints.pdf
    fig06_approval_pipeline.pdf
    fig07_wetlab_boundary.pdf
    fig08_replay_validation.pdf
```

Public bundles must exclude raw sequences, private salts, private approval logs, provider packet internals, and unreviewed paper drafts.

D Implementation Map

Subsystem	**Primary implementation paths**
Cycle runner	`infra/loop_runner.py`
Runtime manifests	`infra/runtime_artifacts.py`
Galen orchestration	`agents/orchestration/galen.py`, `state_machine.py`, `state_writers.py`
Workflow guard	`agents/orchestration/workflows.py`, `workflow_guard.py`
Governance policy	`agents/governance/policies.py`, `runtime_policies.py`
Approvals and remediation	`integrations/telegram/approvals.py`, `approval_consumer.py`, `remediation.py`, `remediation_executors.py`
Telemetry	`agents/telemetry/operations.py`
Evidence retrieval	`pipelines/evidence_index.py`, `retrieval.py`, `evidence_reranker.py`, `research_retrieval.py`
Claim QA and contradictions	`pipelines/claim_qa.py`, `contradiction_graph.py`
Hypotheses and planning	`pipelines/hypothesis_engine.py`, `research_planner.py`, `experiment_planner.py`
Scientific memory	`pipelines/scientific_memory.py`, `memory_consolidation.py`
Provenance graph and IDs	`pipelines/provenance_graph.py`, `onchain_ids.py`, `contracts/v2/*.sol`
Wetlab interfaces	`pipelines/wetlab/interfaces.py`, `agents/wetlab/agent.py`

References

[1] Peter Amstutz, Michael R. Crusoe, Nebojsa Tijanic, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, John Kern, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common workflow language, v1.0. https://www.commonwl.org/, 2016.

[2] Benjamin Burger, Phillip M. Maffettone, Vladimir V. Gusev, Catherine M. Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M. Alston, Buyi Li, Rob Clowes, Nicola Rankin, Brandon Harris, Reiner S. Sprick, and Andrew I. Cooper. A mobile robotic chemist. *Nature*, 583:237–241, 2020.

[3] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35:316–319, 2017.

[4] Anna Gaulton, Louisa J. Bellis, A. Patrícia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40(D1):D1100–D1107, 2012.

[5] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[6] Yaroslav O. Halchenko, Kyle Meyer, Benjamin Poldrack, Dorian Solanky, Adina S. Wagner, Jason Gors, Dave MacFarlane, Dorian Pustina, Vanessa Sochat, Satrajit S. Ghosh, Sören Möller, Russell A. Poldrack, and Michael Hanke. DataLad: distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software*, 6(63):3262, 2021.

[7] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of EMNLP-IJCNLP*, 2019.

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zidek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583–589, 2021.

[9] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP*, 2020.

[10] Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The automation of science. *Science*, 324(5923):85–89, 2009.

[11] Johannes Koster and Sven Rahmann. Snakemake: a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[12] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, 1997.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020.

[14] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

[15] Bertram Ludaescher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[16] Luc Moreau and Paolo Missier. PROV-DM: The PROV data model. W3C Recommendation, 2013.

[17] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[18] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021.

[19] Stian Soiland-Reyes, Peter Sefton, Mercè Crosas, Leyla Jael Castro, Frederik Coppens, José M. Fernández, Daniel Garijo, Björn Grüning, Marco La Rosa, Simone Leo, and others. Packaging research artefacts with RO-Crate. *Data Science*, 5(2):97–138, 2022.

[20] The UniProt Consortium. UniProt: the universal protein knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, 2023.

[21] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proceedings of EMNLP*, 2020.

[22] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016.

[23] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, 41(4):39–45, 2018.