ClaimLedger: Verifier-Governed Claim Maintenance for AI-Assisted Research

David Elliman
Neuro-symbolic Ltd
dave@neusym.ai

2026-07-20

Abstract

Large language models can generate plausible research claims faster than a human can check the claims, their dependencies, and the evidence cited for them. The resulting failure is not only factual hallucination: it is *state corruption* in a living research corpus—confidence is promoted without support, retired results reappear, and scripts that merely print a number acquire the status of tests. We present *Verifier-Governed Claim Maintenance* (VGCM), an architecture in which a generator may propose claim-state transitions but cannot apply a promotion to the authoritative record. We formalise a ledger $\mathcal{L} = (C, D, B, T, S, H)$ comprising stable claims, load-bearing dependency edges, evidence bindings, a corpus-defined confidence order, lifecycle state, and an event history. Six executable invariants govern tier support, retraction propagation, evidence admissibility, reference integrity, history/state consistency, and principal authority. The deterministic audit core is separated from evidence execution and from the human commit decision. We implement the architecture in ClaimLedger, a pure-standard-library Python package with corpus adapters and policy hooks. Eight seeded fault operators are all detected by their assigned verifier layers and a clean fixture produces no findings; these are mechanism-coverage results, not field estimates of precision or recall. A non-physics experiment-log adapter, a measured scaling study, and a parity audit against a maintained physics corpus identify both the reusable kernel and a load-bearing adapter requirement: dependency edges must encode support, not merely citation. The system enforces *consistency, not truth*. Its contribution is a reviewable claim-state machine that remains useful when the generator is unreliable.

Classification. arXiv: primary `cs.AI`; cross-list `cs.SE`, `cs.DL`, `cs.LO`.

Artifact. Source, tests, benchmark data, examples, and this manuscript are public at https://github.com/dgedge/claimledger. Reported results target the tagged release `claimledger` v0.1.0, archived at `doi:10.5281/zenodo.21468895` (concept DOI for all versions: `10.5281/zenodo.21468894`).

1 Problem statement and threat model

Research agents can now propose ideas, write code, execute experiments, and draft papers end to end [7, 8, 9]. That capability does not make the resulting justification chain reliable. Models trained with human feedback can favor agreement with a user's stated view over correctness [5], and intrinsic self-correction without external feedback is unreliable on reasoning tasks [6]. In a long-lived research project, the damaging unit is therefore not one bad answer but an uncontrolled transition in the corpus's *claim state*.

The defended asset is a record of what is currently asserted, at what confidence, resting on which claims, bound to which artifacts, and what has been withdrawn. The primary adversary is a fluent but non-malicious generator. The human is a second-order adversary against themselves

	generator or workflow failure	observed form in the development record	control
A1	confabulated justification	a cited theorem did not exist	I4 references
A2	evidence theatre	an exit-0 script printed but gated nothing	I3 assertion class
A3	retraction resurrection	a retired value survived in live prose	I2 propagation
A4	partial retraction	one leg of a multipart claim was missed	I2 coverage
A5	confidence inflation	a downgraded dependency still supported a high-tier claim	I1 tiers
A6	post-hoc coincidence	a numerical match omitted its search space	policy hook
A7	silent overwrite	failure was edited away rather than recorded	I5 replay
A8	mutual confirmation	desired conclusion and agreeable generator reinforced each other	I6 authority

Table 1: Threat classes that motivated the architecture. The benchmark instantiates one operator per row except A6, which is a domain policy rather than a kernel invariant.

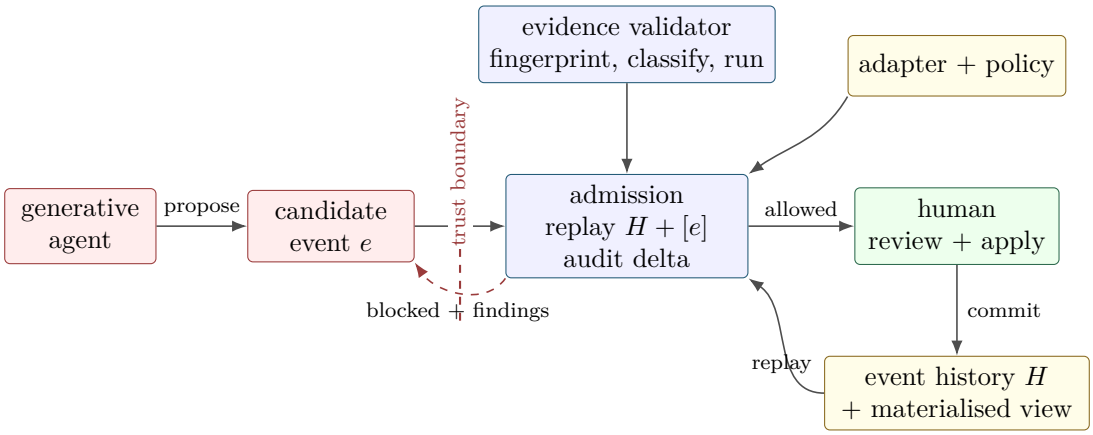

Figure 1: The governed live path. The generator can construct events but has no apply capability. Corpus bootstrap is a separate trusted import path and is not shown.

through motivated reasoning, fatigue, and incomplete propagation of revisions. A malicious operator with arbitrary repository-write access is outside the v0.1 threat model (Section 7).

The architectural rule is:

> **Generators may propose claim-state transitions; a verifier checks admissibility, and only a human-controlled commit path applies an admitted transition.**

Figure 1 makes the boundary explicit. "Deterministic" applies to replay and audit given a ledger, policy, adapter view, and evidence-status cache. Executing an evidence script is a separate operation inside the trust boundary; a nondeterministic or malicious script is not made trustworthy by being launched from ClaimLedger.

Our contributions are: (i) a claim-ledger state model with explicit edge semantics; (ii) six invariants and an admission algorithm; (iii) a generic adapter/policy boundary; (iv) a tested reference implementation; and (v) executable evaluation artifacts that include both positive results and a documented adapter-parity failure. Companion drafts describe the originating manual discipline [1] and its domain-specific PTMS prototype [2]; this paper isolates the reusable

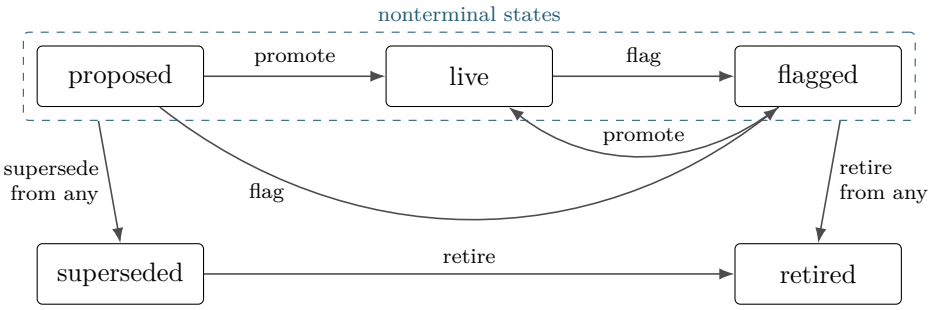

Figure 2: Legal lifecycle transitions. "From any" denotes any of the three nonterminal states. There is no resurrection edge from retired.

computer-science object.

2 Claim-ledger semantics

A ledger is

$$\mathcal{L} = (C, D, B, T, S, H).$$

C is a set of claims with permanent identifiers. $D \subseteq C \times C$ is a directed support relation: $(c, d) \in D$ means that claim c *requires* claim d at the stated level of confidence. This is stronger than citation. A background reference, comparison, or historical note must not be inserted into D unless failure of the cited claim would weaken c. The TCH parity audit in Section 6.3 demonstrates why this distinction is load-bearing.

B binds claims to evidence artifacts. $T : C \rightharpoonup \Lambda$ gives each claim a stated confidence tier. In v0.1, Λ is a finite total order (implemented by a class named `TierLattice`); partial orders are future work. Tier names are supplied by corpus policy, not by the kernel. For example, the bundled corpora use

$$\mathsf{open} < \mathsf{proposition} < \mathsf{locked} \quad \text{and} \quad \mathsf{reported} < \mathsf{reproduced} < \mathsf{replicated}.$$

S assigns lifecycle state, and $H = (e_1, \ldots, e_n)$ is the event sequence.

State is a fold. The materialised claim and evidence dictionaries are a cache of replay(H). Sequence numbers are consecutive and timestamps are caller-supplied, so replay does not read a clock. This gives deterministic state reconstruction and catches a state-only mutation. It is *not* cryptographic tamper evidence: an operator who rewrites both a complete log and its view consistently is outside the model.

Lifecycle. Figure 2 is the transition relation implemented in `events.TRANSITIONS`. Retirement has no outgoing edge. Reconsidering a retired idea therefore creates a new claim and a supersession link rather than erasing the earlier failure.

Principals and authority. Each event names generator, verifier, or human. The generator may propose claims, attach candidate evidence, add dependencies, and annotate. The verifier may flag and annotate but cannot promote. Human events may express every transition, while the live application exposes them through admission. The source-corpus importer is intentionally separate: it bootstraps a reviewed corpus as human-principal events. Python module access is not an access-control system, so deployments must keep direct bootstrap APIs away from the generator.

Evidence state. An evidence binding has an authoritative kind and locator plus verifier-owned derived fields: SHA-256 fingerprint, assertion class, validation status, and a detail string. Executables classify as asserting, check, or silent. Read-only `check` confirms structure and class without execution; `verify` runs executables in separate subprocesses with a timeout. For an evidence-required tier, verification and admission require at least one passing binding; executable evidence must be asserting or checking. External pins and attestations remain unvalidated in v0.1.

3 Safety invariants and admission

The audit emits itemised findings (`check_id`, `severity`, `claim`, `location`, `message`), ordered HARD before SOFT. No aggregate quality score is produced. The CLI exit status is nonzero exactly when at least one HARD finding exists.

I1 — tier soundness. Let $D(c) = \{d : (c, d) \in D\}$. Effective tier is the greatest fixed point below the stated assignment:

$$T_{\text{eff}}(c) = T_{\text{stated}}(c) \wedge \bigwedge_{d \in D(c)} T_{\text{eff}}(d),$$

where the empty meet is lattice top. Unknown tiers and, under the default fail-closed policy, retired or superseded dependencies contribute bottom. A live claim with $T_{\text{eff}}(c) < T_{\text{stated}}(c)$ is HARD. Iteration terminates because meet is monotone decreasing over a finite order; the current implementation propagates along at most $|C| - 1$ edges and has worst-case $O(|C||D|)$ time.

I2 — retraction propagation. A retired claim registers literal signature strings that identify its load-bearing residue in prose. A retirement declaring K independent legs must register at least K signatures. An adapter scans occurrences and maps each source location to an owning claim. An unmarked survival in live text is HARD; a survival in a retraction-aware region without a local marker can be policy-demoted to SOFT. Literal matching is intentionally conservative and incomplete.

I3 — evidence admissibility. Missing and failing artifacts are HARD on live claims; stale artifacts are surfaced. Evidence-required tiers need a structurally admissible binding in read-only mode and a passing binding in verification or admission. A silent executable cannot supply that support. This separates "a file was cited" from "the cited file gated its exit status and passed."

I4 — reference integrity. Every claim dependency and supersession link resolves, every evidence binding targets an existing claim, and corpus-local evidence locators resolve when a root is available. I4 does not decide whether a cited paper entails the claim; semantic scientific-claim verification is a different task [10, 11].

I5 — history/state consistency. H is consecutive and the materialised state equals replay. The invariant detects state mutation around the event API and malformed sequences. It should not be described as a transparency log or as proof that history was never rewritten.

I6 — authority. Every recorded event is rechecked against the principal/event matrix even though `Ledger.apply` already enforces it. This catches an event appended around the normal method when its principal is unauthorized; it cannot detect a forged human identity without an external authentication layer.

Admission algorithm. Given current ledger L and candidate event e, admission computes the set of baseline HARD findings, replays $H + [e]$, carries forward verifier-owned validation cache entries for pre-existing evidence bindings, re-audits, and returns allowed iff the event is legal and introduces no new HARD finding. The baseline-delta rule permits incremental migration of a corpus that already has findings; a strict deployment can additionally require an empty baseline. Admission mutates nothing. A compromised verifier cannot apply an event by itself, but it can incorrectly admit one that a human then applies; verifier integrity remains trusted.

3.1 Worked trace: why the graph matters

Figure 3 shows a minimal example. C_3 has a passing test, but depends on C_2, which depends on open claim C_1. Evidence attached to C_3 cannot compensate for the unsupported dependency: I1 lowers both effective tiers. If C_1 is later retired, its effective contribution becomes bottom, I2 searches for its registered signature in live prose, and the contamination view identifies downstream claims for review. The operations are separate because they answer separate questions: graph support, textual propagation, and evidence execution.

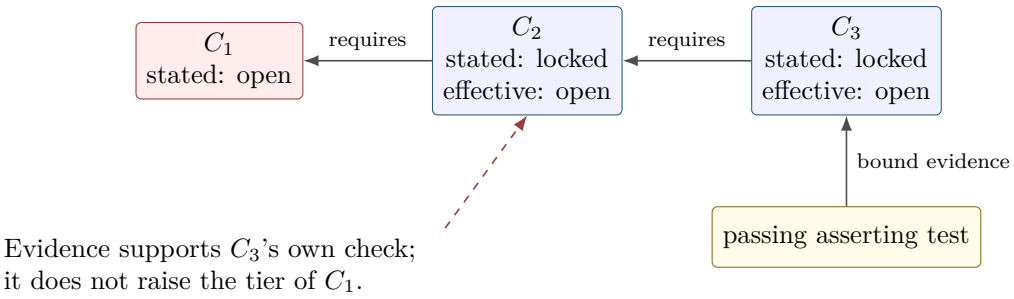

Figure 3: Tier support propagates over load-bearing dependencies. A passing artifact at the leaf cannot repair an open premise upstream.

4 Kernel, adapters, and policies

The original PTMS prototype embedded its domain grammar directly. In ClaimLedger, the kernel sees generic claims, events, bindings, and occurrences. A corpus adapter supplies four methods:

`extract()`	parse claims and evidence bindings
`occurrences(signature)`	find candidate textual survivals
`owner_of(file,line)`	map a location to a claim
`claim_text(claim_id)`	return the owning region for awareness checks

The adapter therefore owns syntax and source-location semantics. It must also preserve the semantic contract of D: extracting every citation as a dependency is incorrect. Policies provide the ordered tier set, evidence-required tiers, contamination strictness, and optional checks. A search-space audit for tolerance-bearing numerical matches is one such domain policy, motivated by multiple-comparisons discipline [17]; it is not part of the kernel.

The repository ships three adapters: a compact Markdown claim microformat, a JSON ML experiment log, and a TCH/PTMS sidecar adapter. The first two are self-contained examples. The third consumes a separately versioned public corpus and is used for the parity analysis below.

5 Reference implementation

ClaimLedger v0.1.0 is a Python 3.10+ package with no runtime dependency outside the standard library. The public repository contains the model, event, ledger, policy, evidence, verifier, and CLI modules; JSON Schemas; three adapters; two example corpora; mutation, scaling, and parity harnesses; and CI across Python 3.10–3.12. The supported commands are `extract`, `check`, `verify`, `events`, and `bench`. Installation for the repository artifact is `pip install -e .` from a checkout; no PyPI publication is claimed.

The package intentionally does not promise a hardened security boundary. `Ledger.emit` exists for bootstrap and tests, object fields are ordinary Python values, executable evidence is unsandboxed, and the JSONL history is not hash-chained. These choices keep the research prototype inspectable; a deployment facing hostile principals should combine it with authenticated service boundaries and artifact attestations.

6 Evaluation

6.1 Seeded-fault mutation benchmark

Eight operators mutate copies of a clean five-claim Markdown corpus. Each operator declares the verifier layer and severity expected to detect it; the full verify-mode audit then runs. Table 2 reports a mutation score of 8/8 and zero findings on the one clean fixture.

operator	seeded failure	expected check	caught
resurrect signature	unpropagated retirement	propagation / hard	✓
drop retirement leg	incomplete multipart retirement	coverage / hard	✓
tier inflate	unsupported confidence	tier soundness / hard	✓
dangling dependency	nonexistent claim reference	references / hard	✓
dangling evidence	missing artifact	evidence / hard	✓
break evidence	executable regression	evidence / hard	✓
silence evidence	exit 0 with no gate	evidence / hard	✓
contaminate	live claim on retired ground	contamination / soft	✓

Table 2: Seeded-fault benchmark. Reproduce with `claimledger bench -ablation`.

This is not a statistical precision/recall evaluation: eight designed mutants and one clean fixture do not define a deployment distribution. The justified claim is narrower: every named mechanism has an executable positive test, the clean fixture is a negative test, and disabling a tested layer makes its assigned mutation(s) invisible. The ablation establishes non-redundancy only within this benchmark.

6.2 Runtime scaling

The committed scaling harness constructs both a shallow graph and a reverse chain that forces tier information to move one edge per sweep. Table 3 reports five-run medians on macOS arm64, Python 3.13.7. The shallow case is approximately linear. The adversarial chain exposes the current $O(|C||D|)$ implementation but remains below two seconds at 3,000 claims. The JSON artifact records platform metadata and min/max timings.

claims	edges	events	shallow (s)	reverse chain (s)
100	99	299	0.00030	0.00238
300	299	899	0.00090	0.02002
1,000	999	2,999	0.00313	0.21396
3,000	2,999	8,999	0.01005	1.87894

Table 3: Audit runtime from `benchmarks/scaling/results.json`. Evidence execution and corpus signature scanning are excluded because their cost is adapter/artifact dependent.

6.3 Longitudinal corpus and adapter parity

The motivating TCH corpus was first committed on 2026-05-16 and evolved under the PTMS prototype. At pinned public commit `3db2fe0` (2026-06-20), its sidecar has 296 claim records and 477 unique cited script paths. The companion PTMS analysis reports 418 assertion-bearing, 27 explicit-check, and 32 pure-print scripts, plus a dependency/contamination worklist [2]. These are PTMS case-study measurements, not results produced by the generic kernel.

The new TCH adapter makes the bridge executable. On public source revision `b5ff087`, it extracts 318 claims, 831 resolved reference edges, and 1,250 claim-to-script bindings with no unresolved reference strings. The result does *not* reproduce the PTMS finding set: it emits 23 tier findings and 82 contamination findings, whereas the PTMS audit has no hard finding and 30 contamination findings. Inspection identifies the principal cause: PTMS `cited_refs` combines contextual citations with load-bearing dependencies, while I1 and contamination interpret every D edge as load-bearing. Two multipart and one evidence finding expose additional status/evidence-policy differences.

This divergence is a result, not a failed demonstration of the kernel. It shows that a general adapter must translate semantics, not only records. The paper therefore does not report historical catch latency: replaying an incorrect edge mapping over Git history would produce precise but invalid numbers. The next parity step is to add an explicit edge role (requires versus context) to the PTMS sidecar, freeze that mapping, and only then run historical replay. The committed parity JSON makes the present boundary reproducible.

6.4 Independent non-physics adapter

The JSON adapter models ML experiment claims under reported $<$ reproduced $<$ replicated. Evidence is an assertion-bearing evaluator pinned to a committed metrics artifact. The adapter is 91 source lines; the kernel and verifier are unchanged. The example audits clean, and a test demonstrates that reusing a retired result after a seed sweep is caught by the same signature-propagation mechanism. This is a proof of interface portability, not evidence that four methods capture every research domain.

7 Limitations and security boundary

Consistency is not truth. A ledger may satisfy every invariant while resting on a false shared premise. Passing code may faithfully implement the wrong model. External replication, forward prediction, expert review, and empirical contact remain necessary.

The dependency graph is curated. I1 is only as meaningful as the distinction between support and context. The TCH parity result demonstrates the failure mode. Automatically inferring edge roles with an LLM would move the original unreliable generator inside a load-bearing verifier input and would need independent validation.

Evidence classification is syntactic. The v0.1 classifier recognizes three Python control forms: assertions, `sys.exit`, and `raise SystemExit`. It confirms that exit status is gated, not that the assertions cover the claim's quoted quantity. It does not inspect imports or prove determinism.

Retraction matching is literal. A paraphrase can evade a signature scan; a broad signature can overmatch. Signatures and multipart counts remain human-curated. Semantic matching is a possible advisory layer, not a replacement for deterministic gates.

No hostile-code sandbox or tamper-proof log. Evidence subprocesses inherit the user's authority and corpus working directory. The event history detects malformed or state-divergent records but has no hash chain, signature, or external witness. Systems facing malicious authors should combine ClaimLedger with a sandbox and a provenance/attestation layer such as in-toto [16]; W3C PROV and RO-Crate provide exchange models for provenance metadata [14, 15] but do not by themselves enforce ClaimLedger's transition policy.

Human and verifier remain trusted. The authority split prevents a generator from applying promotions. It does not make a mistaken human decision or compromised verifier harmless. The value is that the decision, evidence, and downstream impact are inspectable and replayable.

8 Related work

Truth maintenance and belief revision. Classical truth-maintenance systems maintain consistency among beliefs and their justifications [3, 4]. VGCM inherits explicit justification structure but changes the operational setting: the producer of new justifications is untrusted, claims have a research lifecycle, and promotion is governed by evidence and authority. It does not attempt general non-monotonic theorem proving.

Claims, arguments, and assurance. Micropublications represent claims, evidence, argument, and challenge in scholarly communication [12]. Assurance cases and Goal Structuring Notation organize claims and supporting evidence for review [13]. ClaimLedger is narrower about argument semantics—a dependency edge does not encode a full warrant—but adds executable lifecycle checks, retraction propagation, and principal-governed transitions. Conversely, it does not establish the completeness or persuasiveness of an assurance case.

Scientific claim verification. SciFact and SciFact-Open evaluate retrieval of literature evidence and support/refute judgments for natural-language scientific claims [10, 11]. ClaimLedger does not perform textual entailment. It assumes stable claim identities and curated support edges, then checks the maintenance of those structures. The approaches are complementary: a claim verification system could propose evidence bindings, but its output should remain untrusted until admitted.

Provenance and artifact integrity. W3C PROV models entities, activities, and agents [14]; RO-Crate packages research artifacts and metadata for reuse [15]; in-toto verifies the authorized steps and materials of a software supply chain [16]. ClaimLedger's distinct object is the lifecycle and confidence state of research claims. A production system should serialize or attest its ledger using these mature provenance mechanisms rather than invent a weaker storage layer.

AI-assisted research. Autonomous research systems and replication benchmarks increase the rate at which candidate research artifacts can be produced [7, 8, 9]. Sycophancy and weak intrinsic self-correction motivate an architecture that does not ask the generator to certify itself [5, 6]. VGCM is not another self-critique prompt or model-based judge; it is a deterministic governance layer over declared claim state, with all semantic judgments left visible.

9 Conclusion

Verifier-Governed Claim Maintenance treats AI-assisted research as a state maintenance problem. Claims receive stable identities, support edges, evidence bindings, tiers, lifecycle states, and recorded transitions. Generators can enlarge the proposed periphery but cannot apply promotions. The verifier enforces six inspectable invariants, and a human remains responsible for the admitted commit.

The evaluation supports a bounded conclusion. The implemented mechanisms catch their seeded faults, the examples instantiate more than one domain, and the verifier is fast at the tested scales. The TCH parity audit also shows where generality can fail: a citation graph is not automatically a dependency graph. That negative result sharpens the framework's extension contract. The system does not determine truth; it makes the evolution of asserted claims explicit enough that failures can be found, propagated, and reviewed rather than fluently forgotten.

References

[1] D. Elliman. *The Rigid Ground Truth Framework: A Protocol for Catching AI Confabulation in Speculative Research*. Working manuscript, 2026. https://doi.org/10.5281/zenodo.20468553.

[2] D. Elliman. *Relocating Trust to the Verifier: A Truth-Maintenance System for an AI-Generated Theory of Everything*. Working manuscript, 2026.

[3] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979. doi:10.1016/0004-3702(79)90008-7.

[4] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986. doi:10.1016/0004-3702(86)90080-9.

[5] M. Sharma et al. Towards understanding sycophancy in language models. arXiv:2310.13548, 2023. https://arxiv.org/abs/2310.13548.

[6] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. https://openreview.net/forum?id=IkmD3fKBPQ.

[7] C. Lu et al. The AI Scientist: Towards fully automated open-ended scientific discovery. arXiv:2408.06292, 2024. https://arxiv.org/abs/2408.06292.

[8] G. Starace et al. PaperBench: Evaluating AI's ability to replicate AI research. arXiv:2504.01848, 2025. https://arxiv.org/abs/2504.01848.

[9] Z. Chen et al. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025.

[10] D. Wadden, S. Lin, K. Lo, L. L. Wang, M. van Zuylen, A. Cohan, and H. Hajishirzi. Fact or fiction: Verifying scientific claims. In *EMNLP*, pages 7534–7550, 2020. doi:10.18653/v1/2020.emnlp-main.609.

[11] D. Wadden, K. Lo, B. Kuehl, A. Cohan, I. Beltagy, L. L. Wang, and H. Hajishirzi. SciFact-Open: Towards open-domain scientific claim verification. In *Findings of EMNLP*, pages 4719–4734, 2022. doi:10.18653/v1/2022.findings-emnlp.347.

[12] T. Clark, P. N. Ciccarese, and C. A. Goble. Micropublications: a semantic model for claims, evidence, arguments and annotations in biomedical communications. *Journal of Biomedical Semantics*, 5:28, 2014. doi:10.1186/2041-1480-5-28.

[13] Y. Matsuno. Design and implementation of GSN patterns: A step toward assurance case language. *Information and Media Technologies*, 9(3):262–271, 2014. doi:10.11185/imt.9.262.

[14] L. Moreau and P. Missier, editors. *PROV-DM: The PROV Data Model*. W3C Recommendation, 30 April 2013. https://www.w3.org/TR/prov-dm/.

[15] S. Soiland-Reyes et al. Packaging research artefacts with RO-Crate. *Data Science*, 5(2):97–138, 2022. doi:10.3233/DS-210053.

[16] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium*, pages 1393–1410, 2019.

[17] A. Gelman and E. Loken. The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition." Technical report, Columbia University, 2013.