Research Square

Preprints are preliminary reports that have not undergone peer review. They should not be considered conclusive, used to inform clinical practice, or referenced by the media as validated information.

From Raw Agent Traces to Auditable Claims: A Reproducibility Layer for AgentDojo

Abdullah Alharbi

albdrani36@gmail.com

Independent Researcher

Method Article

Keywords: agent evaluation, research software, provenance, reproducibility, benchmark auditing, fault localization

Posted Date: October 16th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11029312/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.

Abstract

Interactive-agent benchmark claims are difficult to audit when source identity, selection, denominators, derived records, and claim locators are disconnected. AgentDojo Trace Audit is a Python layer for a fixed AgentDojo v1.2.1 snapshot. It reconstructed 11,624/11,624 selected records, 11,624/11,624 raw-to-derived links, 11,236 attack-present and 388 no-injection records, the four declared outcome cells, and official-evaluator agreement for 11,624/11,624 records. A preserved first confirmatory catalog detected and classified 408/408 mutations but localized 336/408; all 72 failures were retained and analyzed. Following repairs to diagnostic-location representation, a post-repair development catalog was run, and a separately preregistered post-repair regression catalog tested 360 fresh operator-site pairs, including 180 compound challenges, achieving 360/360 detection, classification, and localization on its one-shot run. Two separately frozen benign suites produced 0/120 false alarms on declared transformations. The package references rather than redistributes AgentDojo source, traces, or model outputs. Results are exact within one snapshot and designed catalogs; external replicability is not established.

1. Motivation and significance

Agent benchmarks increasingly evaluate systems that act through tools, modify application state, and encounter adversarial instructions. A final score can be useful, but it does not by itself reveal which executions were selected, whether exclusions were applied consistently, how denominators were formed, or whether a manuscript number can be reached from the underlying trace. These questions become especially important when a benchmark has multiple run groups, source versions, missing records, or source-native outcome labels. An audit should make such decisions executable rather than leave them as prose.

AgentDojo is a dynamic environment for evaluating prompt-injection attacks and defenses for tool-using agents [1]. Its publication and official repository supply the benchmark, tasks, evaluator, metrics, and run material on which this work depends. The software presented here does not replace AgentDojo, republish its traces, or claim ownership of its evaluation design. Instead, it addresses a narrower reproducibility problem: binding one frozen source snapshot to file identities, record-selection decisions, population membership, denominators, derived rows, reported results, and claim locators in a single inspectable path.

This distinction matters because several adjacent systems already solve broader provenance and reproducibility problems. PROV-O supplies a general interoperable provenance ontology [4]. RO-Crate packages research objects and machine-readable metadata [5]. ReproZip captures execution dependencies for replay [6], while DataLad versions code, data, and process relations [7]. SPDX and SLSA address software composition and build provenance [8, 9]; Citation File Format and CodeMeta standardize citation and software metadata [10, 11]. The FAIR for Research Software principles frame findability, accessibility, interoperability, and reuse [12]. AgentDojo Trace Audit uses or aligns with these

capabilities where appropriate, but does not present generic provenance, packaging, citation metadata, software bills of materials, or build attestations as inventions.

The closest scientific comparison is Gao and Zhou's outcome-evidence layer, which assigns evidence labels and reports score bounds across five agent benchmarks, including AgentDojo [2]. This work instead binds a fixed source content identity to executable selection, denominator, lineage, result, and claim-target checks. The questions overlap but the methods and outputs differ. A current survey places claim support, tool outputs, actions, and final answers within a broader evidence-tracing and execution-provenance field [3]. The contribution here is therefore a benchmark-specific executable audit path, not a priority claim for provenance or reproducibility concepts.

The primary research question is: can a trace-linked, denominator-frozen software layer reconstruct and audit a fixed AgentDojo corpus without redistributing raw traces, while detecting and localizing declared provenance failures? The software is intended for benchmark authors, agent-evaluation researchers, auditors, and reproducibility reviewers who already possess a lawful local copy of the official source prerequisite. It is not a safety classifier, causal estimator, replacement benchmark, or general trace-observability platform.

2. Software description

2.1 Design goals and architecture

The architecture separates a generic validation core from an AgentDojo adapter. The generic core checks source-file identities, unique unit identifiers, relative paths, population membership, selection bijection, denominator closure, lineage edges, result values, and claim-target hashes. The adapter reads AgentDojo source-native JSON fields and maps them to a stable record schema. Command-line stages expose preparation, analysis, validation, rendering, and regression verification. Expected historical counts are used only after reconstruction in a declared regression receipt; they are not inputs to record selection or derivation.

The end-to-end path begins with source acquisition. A user obtains the official AgentDojo repository at commit 089ed468cf3ed0322acc66b0211f26d9d90dbf60, corresponding to benchmark version v1.2.1, or an official archive of that commit. The prepare stage safely extracts an archive, rejects traversal or link members, locates the source root, computes every used file hash, and produces a normalized content-tree identifier. This identifier is computed from sorted records containing file kind, relative path, byte size, and SHA-256. It is deliberately independent of archive member order, compression, timestamps, and the outer directory name.

The distinction between SOURCE_CONTENT_ID and SOURCE_CONTAINER_SHA256 prevents a benign container change from being mislabeled as source drift. The exact historical archive remains identified by its container hash, which proves which bytes were used in this study. A Git checkout, GitHub source archive, or user-created archive is accepted only when its normalized content tree matches the frozen

source content. Tests confirmed that a differently ordered, differently named archive produced the same content identifier, while an isolated byte change produced a different identifier and a localized file mismatch.

After source verification, analyze scans four frozen run groups and applies one explicit selection rule. A primary record must use benchmark v1.2.1, have a user_task_* identifier, and contain non-null utility, security, and duration fields. Each included record receives a unit identifier derived from its source-relative path, the raw-file SHA-256, the full source commit, benchmark version, run group, population identifier, and source-native outcomes. Records are sorted by UTF-8 relative path before serialization. This canonical order removes irrelevant enumeration differences without discarding content.

The validate stage reopens each selected source file and recomputes its hash. It checks that selected paths and unit identifiers are unique; paths are relative; all records use the frozen commit, benchmark version, and population; the attack-present and no-injection denominators partition selected records; the four attack-present cells partition that denominator; and every selected record has one raw-to-derived lineage entry. Claim locators identify a result file and JSON pointer, and a target hash binds the locator to exact result bytes. Structured diagnostics name the affected file and, when the invariant permits, the field.

The render stage creates a human-readable reconstruction report. verify compares reconstructed outputs with the frozen demonstration receipt only after analysis is complete. run_all composes preparation, analysis, validation, and rendering; verify_all performs regression verification and automated tests. Exit codes distinguish command misuse from scientific or input validation failures. Runtime operation uses the Python standard library and makes no network request.

2.2 Population accounting and the incomplete record

The four frozen run groups contain 11,765 candidate JSON files. The executable selection register assigns an INCLUDE or EXCLUDE decision and reason to every candidate. It excludes 140 auxiliary injection_as_user_task cases because their identifiers do not represent the primary user_task_* population. One primary record is incomplete: runs/meta-llama_Llama-3.3-70B-Instruct-repeat_user_prompt/banking/user_task_10/important_instructions/injection_task_7.json lacks utility, security, and duration. Applying the field-completeness rule gives 11,765 − 140−1 = 11,624 selected records.

The incomplete record is not hidden from the flow. Its exact path and missing fields are registered. Because the outcome fields are absent, the record cannot be assigned to a task-success or attack-goal cell without invention. A sensitivity analysis places it counterfactually in each of the four logically possible attack-present cells. Every placement increases the selected and attack-present denominators by one and increases one cell by one. This demonstrates the numerical effect of alternative placements, but does not identify which placement is true. The rule therefore preserves a reproducible population boundary without interpreting the missing outcome.

The public software contains a small synthetic negative-control fixture for population and mutation checks. It does not include or combine a separate study population. Population identifiers are guarded, and any cross-population relation causes a safe failure.

2.3 Distribution boundary

The public release includes owner-authored source code, schemas, tests, documentation, metadata, a synthetic fixture, and aggregate validation receipts. It does not contain AgentDojo source code, raw traces, or model outputs. A user supplies the official source locally. The software verifies that prerequisite by commit, version, file manifest, normalized content identity, and per-record hashes.

This reference-only design serves two purposes. First, it keeps the audit executable: a lawful source holder can reproduce the selection, denominators, derived records, and diagnostics. Second, it avoids treating the upstream repository license as an automatic grant for every collected trace or model output. The item-level release manifest records origin, rights status, license, third-party content, personal-data or secret status, inclusion decision, and evidence for each included file. LICENSE, NOTICE, third-party notices, citation metadata, CodeMeta, RO-Crate metadata, and an SPDX 2.3 software bill of materials are included. These records are engineering controls, not legal advice.

2.4 AI-assisted development and verification

OpenAI ChatGPT and Codex materially assisted literature search, planning, drafting, code and test development, mutation-catalog and oracle implementation, result recomputation, metadata, programmatic figures, and byte-level checks. Expected outcomes were frozen outside the validator before Catalog D and the 100-control suite were executed; AI-generated suggestions were treated as untrusted and checked against executable receipts, one-shot chronology, and primary sources. No complete model-version identifier was retained, so none is asserted. The author remained the sole scientific decision-maker, reviewed and approved the public release and final manuscript, and retains full responsibility for the submitted article.

3. Illustrative reconstruction and validation

3.1 Frozen corpus reconstruction

The full run recovered 11,624/11,624 selected records. Each selected record had a resolvable raw source file and matching SHA-256, yielding 11,624/11,624 raw-to-derived links. The attack-present denominator was 11,236 and the no-injection denominator was 388, which sum to 11,624. Within attack-present records, the declared orientation produced 6,893 task-success/attack-not-achieved records, 200 task-success/attack-achieved records, 3,727 task-failure/attack-not-achieved records, and 416 task-failure/attack-achieved records. These four cells sum to 11,236.

Table 1
Frozen attack-present cells (n = 11,236), AgentDojo v1.2.1.

Task outcome	Attack not achieved	Attack achieved
Success	6,893	200
Failure	3,727	416

The table is a deterministic description of the frozen corpus and a reconstruction target, not a new safety metric. utility and security are source-native labels. Replaying the official evaluator agreed for 11,624/11,624 selected records, which establishes agreement with that evaluator on this snapshot. It does not establish independent construct validity, real-world safety, or the absence of evaluator error. No statistical confidence interval or superpopulation inference is attached to exact finite-corpus counts.

Four historical acceptance fixtures - missing file, byte drift, version mixing, and denominator corruption - were detected and localized with zero false success. This establishes behavior for those four fixtures only; the larger catalogs below provide separate, explicitly bounded evidence.

3.2 Development and confirmatory mutation catalogs

Development/Conformance Catalog A first exposed 16 localization deficiencies. After the verifier was repaired, rerunning the same 229-case catalog yielded 229/229 post-repair conformance. Because A informed repair, this is development evidence rather than an untouched estimate.

Catalog B comprised 24 synthetic units crossed with 17 operators, giving 408 eligible cases frozen before its one-shot run. It detected and correctly classified 408/408 mutations, but only 336/408 met the frozen localization criterion (82.4%). The 72 failures remained in the original denominator. Forensic review assigned every failure to one of four operator families: 24 lineage deletions, 24 lineage-edge swaps, 12 source swaps without label updates, and 12 Boolean-as-integer cases. The failures reflected diagnostic-location representation rather than missed detection or classification. Catalog B remains the preserved confirmatory partial result.

The repair changed diagnostic representation at the invariant level: record locations were decomposed into file, unit, and terminal-field components; missing-lineage diagnostics used a field-qualified unit locator; and claim targets were checked by resolving JSON pointers. It did not encode the 72 failed case identifiers or remove any case from Catalog B.

Catalog C crossed 30 synthetic units with 11 operators and achieved 330/330 detection, classification, and localization after repairs. It is explicitly a post-repair development catalog and does not replace Catalog B.

Catalog D was a preregistered post-repair confirmatory regression challenge, not a naive or independent holdout. Before execution, the verifier commit, dependency state, seed 9029, generator, oracle, 30 fresh synthetic units, 12 operators, 360 eligible operator-site pairs, and success criterion were frozen. Four of

its 12 operator families (120/360 cases) directly revisited failure mechanisms seen in Catalog B: Boolean type confusion, lineage deletion, lineage redirection, and source swapping. Eight families (240/360 cases) did not directly revisit those failed mechanisms. Half the cases (180/360) were compound challenges, including coordinated raw/manifest changes, source swaps with surface metadata updated, result-to-claim drift, constant-count record replacement, and cross-field inconsistency. The generator and oracle did not import the verifier; this module-level separation prevents direct runtime leakage but is not conceptual independence because the verifier, generators, and oracles share authorship and an AI-assisted development environment. The one-shot run detected 360/360, classified 360/360, and localized 360/360; there were no survivors and no post-run change to the denominator or criterion.

A frozen primary suite exercised 100 eligible benign transformations across 10 families, including archive naming, compression, order, timestamps, permissions, wrapper directories, comments, invocation paths, and non-scientific optional metadata serialization. A separately frozen 20-control supplement covered line-ending changes in optional metadata and NFC/NFD variants in outer wrapper names. The two designed suites produced 0/120 false alarms; this exact count is not a population-wide false-alarm estimate. Applied inside identity-bound source bytes or verified relative paths, line-ending or Unicode-normalization changes are intentionally treated as identity changes and fail closed. Portability for Unicode-equivalent inner paths is therefore not established.

Table 2
Preserved development, confirmatory-regression, and benign-control results.

Catalog or suite	Evidential role	Construction	Observed outcome
B	Preserved confirmatory	24 units × 17 operators = 408	408/408 detect/classify; 336/408 localize
C	Post-repair development	30 units × 11 operators = 330	330/330 in all arms
D	Post-repair confirmatory regression	30 units × 12 operators = 360	360/360 in all arms
Benign controls	Two frozen designed suites	100 + 20 = 120	0/120 alarms

Table 3
Relationship of Catalog D operator families to preserved Catalog B localization failures.

Relationship to Catalog B failures	D operator families	D cases
Directly revisits a failed mechanism	4/12	120/360
Does not directly revisit a failed mechanism	8/12	240/360

3.3 Reimplementation evidence

Two extraction paths produced the same selected-record count, denominators, and four cells. Their functions are in separate modules and neither reads stored governing counts during extraction. A changed synthetic input also changed the derived output rather than being forced back to historical values, reducing one circularity risk.

The paths share the source, selection specification, author, governance, and AI-assisted development environment. They are described as a specification-based reimplementation under shared authorship and governance. This reduces one implementation-specific circularity risk but does not remove correlated specification error and is not external replication.

3.4 Performance and user journey

Performance was remeasured on Linux 6.18.35 with CPython 3.12.13. After one unreported warm-up, ten measured full reconstructions had a median wall time of 2.676231 s (IQR 0.634 s, range 2.445–3.564 s), corresponding to 4,343 records/s, with maximum observed peak resident memory of 46,352 KiB. Validation and receipt generation had a median of 0.595830 s (IQR 0.124 s, range 0.572–1.170 s), corresponding to 19,509 records/s.

Tree discovery and content hashing were measured separately over 11,765 candidates for ten repetitions. Median discovery time was 0.204 s and median hashing time was 0.367 s. For retained-record scaling, discovery was performed once and fixed; transformation was then measured ten times at each size. Median times were 0.056, 0.150, 0.292, 0.456, and 0.682 s at 1,162, 2,906, 5,812, 8,718, and 11,624 records. These separately timed microbenchmarks are not an additive decomposition of the 2.676231 s full reconstruction: they exclude JSON parsing, selection and exclusion-register construction, identifier and record construction, population aggregation, canonical sorting, and output serialization and writes. The measurements support an observed stage-specific throughput description in the tested environment, not a theoretical complexity claim.

A clean wheel installation, command help, a complete reconstruction, and verify_all succeeded on CPython 3.12.13 on Linux. The packaged unittest harness contains 11 tests; mutation catalogs, benign-control suites, full-corpus reconstruction, clean-path checks, and historical-regression checks run as separate scripted workflows. The public CI additionally runs source-level compilation and unit tests on Python 3.11, but the v1.0.0 package metadata supports only Python 3.12; neither wheel installation nor full-corpus execution is claimed for Python 3.11 or 3.13. Runtime commands require no credential or network call. Missing source, content drift, wrong version, denominator drift, duplicate identity, and population mixing fail closed with structured diagnostics.

3.5 Failure semantics and maintainability

Failure behavior is part of the research result because a provenance tool that exits successfully after corruption can be more misleading than a tool that is unavailable. Diagnostics therefore distinguish four classes that were conflated in the earlier implementation. A container-only difference reports that the supplied archive bytes differ while the normalized scientific content remains equivalent. A content-drift

failure reports the relative file whose size or hash changed. A version failure compares the declared commit and benchmark version with the frozen acquisition receipt. A semantic failure identifies the violated selection, population, denominator, lineage, result, or claim invariant. These messages are machine-readable and also suitable for a human log.

Idempotence and ordering receive separate treatment. The extractor sorts selected records by source-relative path before serialization, so input enumeration order does not alter canonical outputs. JSON key order and benign metadata outside the scientific identity are ignored by semantic validation. In contrast, path changes, duplicate identities, altered selection membership, or redirected lineage are not normalized away because they change the research object being audited. This distinction lets the package tolerate harmless transport differences without weakening checks on scientifically relevant identity.

The generic core implements benchmark-agnostic invariants, while the adapter maps AgentDojo JSON to audit records. Schemas, synthetic fixtures, stable CLI stages, and separated unit, mutation, negative-control, clean-path, and regression workflows make failures inspectable. A changelog records behavioral changes; a local build attestation records release inputs and commands without claiming a SLSA level.

Maintenance is conservative: a future AgentDojo commit is not automatically equivalent to v1.2.1. Another version requires an explicit adapter or mapping receipt, selection and denominator rules, and new expected behaviors. Another benchmark likewise requires its own population, source-native fields, missingness rules, and claim boundaries. The generic core may be reused, but benchmark-specific semantics may not be transferred by analogy.

4. Impact, reuse, and comparison

The software adds a review surface between official benchmark material and a manuscript. A benchmark author can use it to freeze selection and denominator rules before reporting. A reviewer can follow a claim locator to a result, derived record, source path, and byte hash. A maintainer can distinguish an archive repack from scientific content drift. An auditor can inject declared faults and inspect whether failures are detected at the expected level. These uses are complementary to AgentDojo's benchmark and evaluator rather than replacements for them.

Compared with Gao and Zhou [2], this package does not estimate evidence-supported score bounds or assign Evidence Pass, Evidence Fail, and Unknown labels across multiple benchmarks. It contributes executable source-content, selection, denominator, lineage, claim-target, and mutation invariants for one snapshot. Compared with PROV-O and RO-Crate [4, 5], it supplies a constrained benchmark profile and acceptance behavior rather than a new ontology or packaging standard. Compared with ReproZip [6], it checks semantic benchmark quantities rather than capturing operating-system dependencies. Compared with DataLad [7], it does not provide general data logistics; compared with SPDX and SLSA [8, 9], it does not claim software supply-chain innovation or a SLSA level. CFF and CodeMeta are used only for correct metadata [10, 11].

Reuse is appropriate when the user has a fixed AgentDojo snapshot and needs reproducible selection, population, denominator, lineage, result, or claim checks. The generic validator may also inform adapters for other benchmarks, but no cross-benchmark adapter or empirical generalization has been validated. The software should not be used to infer real-world harm, certify an agent as safe, estimate causal effects of attacks or defenses, validate benchmark constructs, or claim that AgentDojo as a whole is correct.

5. Limitations

The demonstration covers one AgentDojo commit, one benchmark version, four run groups, and a finite corpus. The adapter and selection semantics are benchmark-specific. Catalog A and Catalog C are development evidence; Catalog B preserves a partial confirmatory localization failure; Catalog D is a preregistered post-repair confirmatory regression challenge; and the benign controls are designed synthetic spaces with frozen one-shot results. Module separation does not establish conceptual independence because the verifier, generators, oracles, and extraction paths share authorship, governance, and an AI-assisted development environment. External replicability is not established. Performance and clean installation were measured only on CPython 3.12.13 on Linux, and portability for Unicode-equivalent inner paths was not tested.

The exclusion sensitivity cannot recover the missing outcome, and the reported counterfactual cells must not be read as imputed observations. Likewise, exact hashes can detect byte changes but cannot establish that an unchanged upstream record is factually or normatively correct. These epistemic limits remain even when every software test passes.

A coherent adversary can alter a source value and update every dependent manifest hash, derived record, summary, and claim target consistently. Because all internal invariants then remain satisfied, the layer cannot detect that change without an external authenticity anchor. The software establishes consistency with supplied trust anchors, not independent authenticity; signed or independently archived source attestations would be required to close this class.

6. Conclusions

AgentDojo Trace Audit binds source content identity, selection, population accounting, denominators, derived records, results, and claim locators into one executable audit chain. It exactly reconstructed 11,624 selected records and the declared finite-corpus results. The validation history remains visible: Catalog B detected and classified 408/408 cases but localized 336/408, exposing 72 confirmatory failures; Catalog C was post-repair development evidence. A preregistered post-repair Catalog D regression challenge then achieved 360/360 detection, classification, and localization, while two separately frozen benign suites produced 0/120 alarms. These are bounded software-validation results under shared authorship and governance, not external replication, construct validation, or general detector performance.

Declarations

Data and code availability

AgentDojo Trace Audit version 1.0.0 is publicly available from GitHub at https://github.com/albdrani37-hash/agentdojo-trace-audit and is archived on Zenodo at https://doi.org/10.5281/zenodo.22217485 [13]. The corresponding GitHub release is https://github.com/albdrani37-hash/agentdojo-trace-audit/releases/tag/v1.0.0; tag v1.0.0 resolves to repository commit 245c9d4fc4ec583c33ccda239b787da399225c28. The version-independent Zenodo concept DOI is https://doi.org/10.5281/zenodo.22217484. AgentDojo source code, raw traces, and model outputs are not redistributed; users must separately obtain the official AgentDojo source at commit 089ed468cf3ed0322acc66b0211f26d9d90dbf60.

Clinical trial number

Clinical trial number: not applicable.

Ethics and consent to participate

Ethics and Consent to Participate declarations: not applicable.

Consent for publication

Consent to Publish declaration: not applicable.

Author contributions

Abdullah Alharbi: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing – original draft, Writing – review and editing, Visualization, Project administration, Resources, and final accountability.

Funding

This research received no external funding.

Competing interests

The author declares no competing interests.

Acknowledgments

The author thanks the AgentDojo authors for releasing the benchmark materials.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During preparation of this work, the author used OpenAI ChatGPT and Codex for literature search, drafting, code and test development, mutation-study design and implementation, recomputation, formatting, metadata, programmatic figures, and byte-level checking. Outputs were checked against primary sources and executable receipts. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication. Research-method use and controls are described in Section 2.4.

Code metadata

C	Description	Public release value
C1	Current code version	1.0.0
C2	Permanent repository	https://github.com/albdrani37-hash/agentdojo-trace-audit
C3	Archival record	https://doi.org/10.5281/zenodo.22217485
C4	License	Apache-2.0 for included owner-authored code
C5	Version control	git; tag v1.0.0; commit 245c9d4fc4ec583c33ccda239b787da399225c28
C6	Language	Python 3.12
C7	Environment	Linux tested; official AgentDojo source prerequisite
C8	Documentation	Repository README, installation guide, and user guide
C9	Support	albdrani36@gmail.com

References

1. Debenedetti E, Zhang J, Balunović M, Beurer-Kellner L, Fischer M, Tramèr F. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. Adv Neural Inf Process Syst. 2024;37:82895–920. https://doi.org/10.52202/079017-2636.
2. Gao S, Zhou L. Can Agent Benchmarks Support Their Scores? Evidence-Supported Bounds for Interactive-Agent Evaluation, arXiv:2605.10448 (2026). https://arxiv.org/abs/2605.10448. Preprint.

3. Wang Y, Zhang J, Cai T, Liu Z, Sun Q, Sun Z, Wu Z, Dong M, Zheng M, Yin X, Zhu Y. From Agent Traces to Trust: A Survey of Evidence Tracing and Execution Provenance in LLM Agents, arXiv:2606.04990 (2026). https://arxiv.org/abs/2606.04990. Preprint.
4. Lebo T, Sahoo S, McGuinness D et al. PROV-O: The PROV Ontology, W3C Recommendation, 30 April 2013. https://www.w3.org/TR/prov-o/
5. Sefton P, Carragáin EÓ, Soiland-Reyes S, et al. RO-Crate Metadata Specification. 2025;1(2). https://doi.org/10.5281/zenodo.13751027.
6. Rampin R, Chirigati F, Shasha D, Freire J, Steeves V. ReproZip: The Reproducibility Packer, Journal of Open Source Software 1(8) (2016) 107. https://doi.org/10.21105/joss.00107
7. Halchenko YO, et al. DataLad: distributed system for joint management of code, data, and their relationship. J Open Source Softw. 2021;6(63):3262. https://doi.org/10.21105/joss.03262.
8. Workgroup SPDX. Software Package Data Exchange (SPDX) Specification 2.3, 2022. https://spdx.github.io/spdx-spec/v2.3/
9. OpenSSF SLSA, community SLSA, Specification. 1.2, 24 November 2025. https://slsa.dev/spec/v1.2/
10. Citation File Format community. Citation File Format 1.2.0. https://citation-file-format.github.io/
11. CodeMeta. community, CodeMeta 3.0. https://codemeta.github.io/
12. Barker M, et al. Introducing the FAIR Principles for research software. Sci Data. 2022;9:622. https://doi.org/10.1038/s41597-022-01710-x.
13. Alharbi A. AgentDojo Trace Audit, Version 1.0.0, Zenodo (2026). https://doi.org/10.5281/zenodo.22217485

Figures

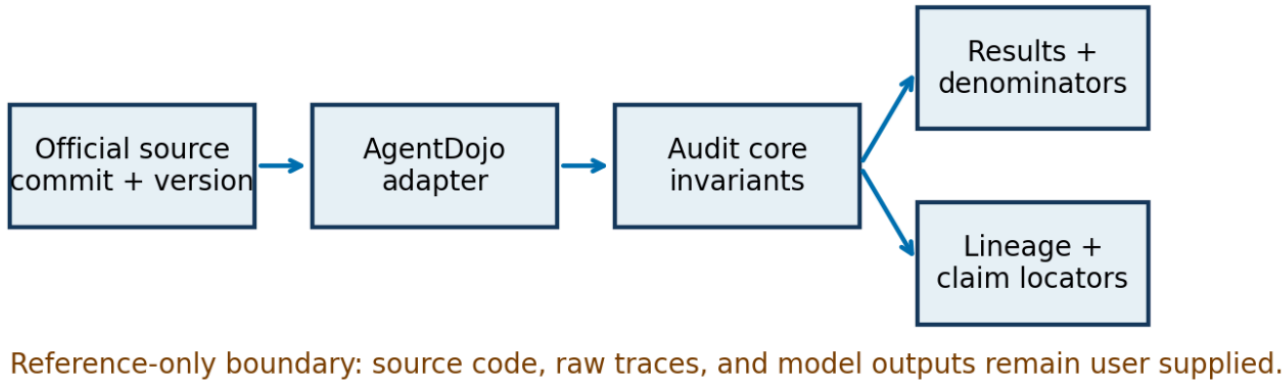
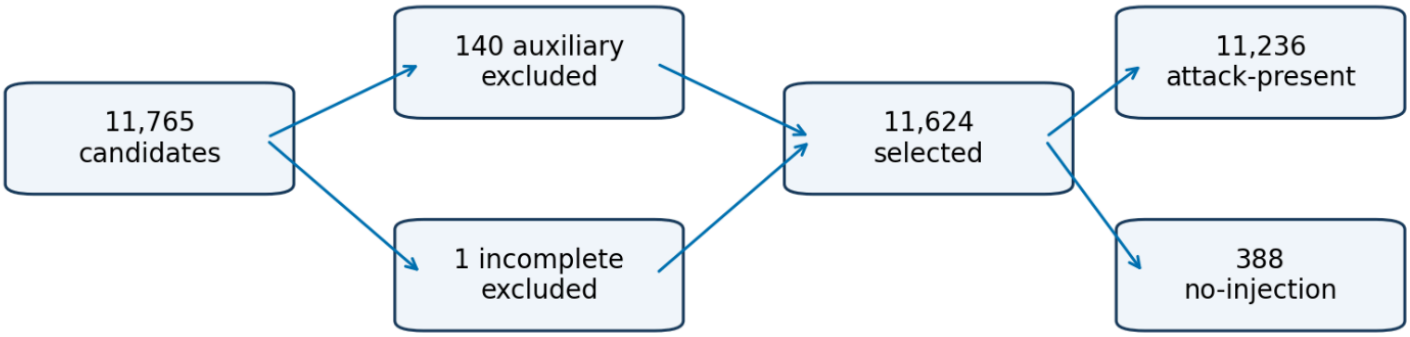


Figure 1

Executable source-to-claim architecture and reference-only distribution boundary.



Executable closure: 11,765 - 140 - 1 = 11,624; 11,236 + 388 = 11,624.

Figure 2

Frozen population accounting. Every candidate receives an executable inclusion decision.

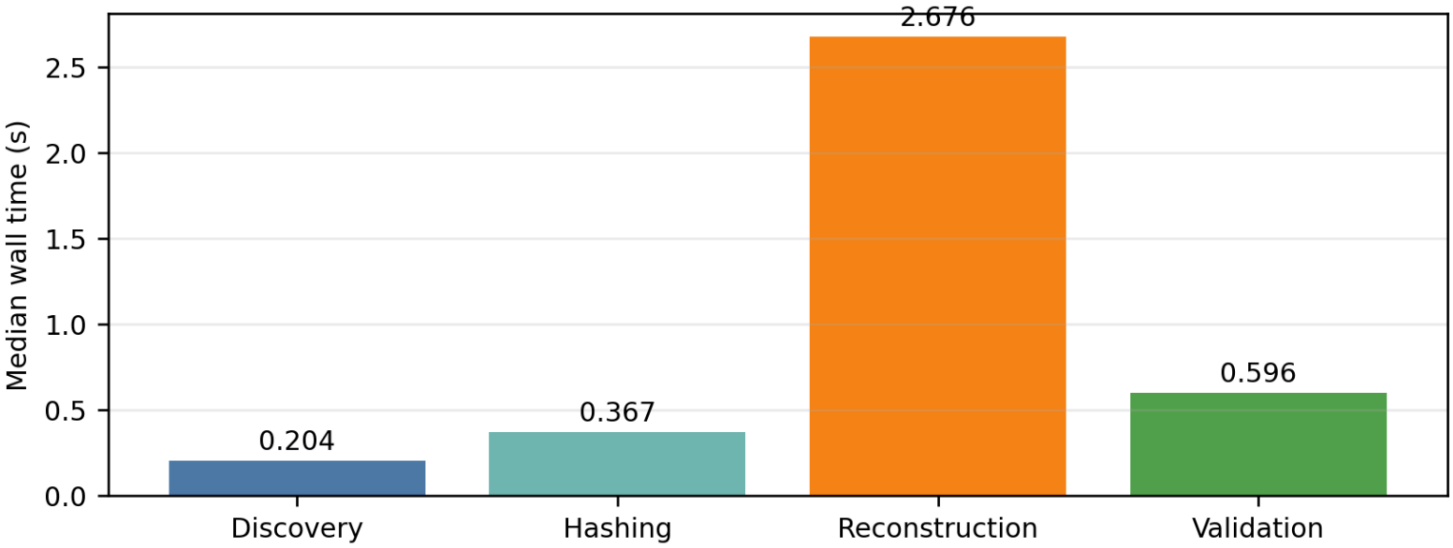


Figure 3

Stage-aware median wall time from ten measured repetitions on Linux.