From a Cold Clone to a Proof Receipt

Agent-native navigation and incremental validation for a large Lean development

Will Cook

July 2026

Abstract

A large formal library can be mechanically exact and still be practically unreadable to the next human or reasoning agent. This paper presents a repository architecture that separates first-contact comprehension from proof checking. At the audited revision, a cold clone exposes a bounded six-line tour, a mathematical map of ten programmes, a reviewed public claim map with five explicit open propositions, and an elaborated loaded-root reference graph joined to source coordinates. These are committed navigation products: they require no Lean build and make omissions and authority boundaries explicit. The inventory behind them spans 1,019 Lean modules and 153,253 declarations; 503 of those modules and 8,171 of those declarations are explicitly marked machine-generated certificate shards, counted as formal source and never as separate mathematical claims. The marked figures are a classification floor, not the generated share: large emitted families predate the markers, so the true machine-generated share is substantially higher.

Navigation does not receive proof authority. A session notary records an agent's observations, falsifiable conjectures, abandoned routes, exact Lean probes, and claims. Probe verdicts come from the pinned Lean process and cannot be authored by the agent; claims must cite an accepted probe, and replay reruns the stored bytes. Compilation is similarly separated from orientation. Lake outputs and content traces support focused or changed-cone builds, while an exact cached receipt prevents an unchanged dependency-index check from repeating a full environment export. A dogfood session records six reasoning notes, one accepted probe, and two claims, and its probe replays at the audited revision.

The contribution is an authority-preserving composition of exhaustive inventory, selective interpretation, bounded intent routing, receipted reasoning, and incremental validation. It is not an autonomous theorem prover, a portability study, or evidence that the agent's reasoning was optimal or mathematically novel outside the repository history.

1 The cold-clone problem

The first interaction with a large Lean repository is usually a filesystem. The reader sees source files, generated files, scripts, papers, and build configuration, but not the shape of the mathematics. A language model can search filenames and text, yet those operations do not answer the questions that matter: Which programmes exist? Which

Authorship and AI use. The prose in this version was generated by large-language-model agents under Will Cook's direction. Cook set the objectives and acceptance criteria, reviewed and authorised the manuscript, and is responsible for its contents. The agents are production tools, not authors. See Appendix A.

propositions are established, conditional, finite, or open? Where does a declaration sit in the formal dependency graph? Which surface is exhaustive, and which is a human selection? What may be trusted without compiling anything?

This is not only a convenience problem. When an agent cannot see a library's option surface, it tends to rediscover existing lemmas, confuse a finite result with its open unbounded neighbour, or build a new local index because the existing one was not discoverable. Conversely, loading every declaration, paper, and proof edge into one prompt destroys the distinctions the extra context was meant to reveal. A useful first-contact surface must therefore be small, expandable, and honest about what it omits.

The development studied here indexes eight open Erdős problems. Problems 249 and 257 are the two principal reviewed programmes; Problems 68, 243, 251, 269, 1041, and 1049 are problem-owned expansion lanes with their own notes and explicit non-claims. The corpus concentrates its depth around a small number of hard frontiers: exact separation equivalences for Problem 249, a machine-checked rational countermodel agreeing with the totient's parity at every index, and quotient-greedy classifications for Problem 257, with obstruction theorems recorded beside the routes they close. The generated certificate shards named in the abstract sit underneath these results as checked finite evidence, not beside them as further claims. The design goal is not to replace reasoning with a fixed pipeline. It is to let a capable reader see enough structure to reason well, while reserving mathematical authority for the pinned proof kernel.

2 A layered mathematical option surface

No single graph can serve all readers without hiding an important distinction. The repository therefore commits four related layers (Figure 1). They are projections with different coverage contracts, not successive claims of understanding.

Exhaustive inventory. The declaration atlas is a source-derived address book. It records every live declaration that its explicit source roots contain. Exhaustiveness here means that a reader can locate a declaration, not that the system understands its mathematical role.

Exact formal neighbourhoods. The dependency index loads the two supported compact roots, extracts direct constant references from elaborated types and values, and joins those constants to atlas coordinates. It reports unresolved rows and edges instead of silently treating absence as independence. Its graph is exact for the loaded environment and stated edge relation, but it is not a transitive proof explanation.

Tiered mathematical interpretation. The semantic graph contains authored statement nodes, typed relations, and an exact source-structural floor. At the audited revision all 143,052 authored theorem-like declarations are linked. Of these, 139,737 (97.7%) participate in authored mathematical interpretations: 3,253 are exact proposition evidence and 136,484 are bounded contextual links to digest- or module-verified families. The remaining 3,315 are grouped only by exact source module and normalised Lean proposition signature. That lower tier is useful navigation, not a mathematical paraphrase.

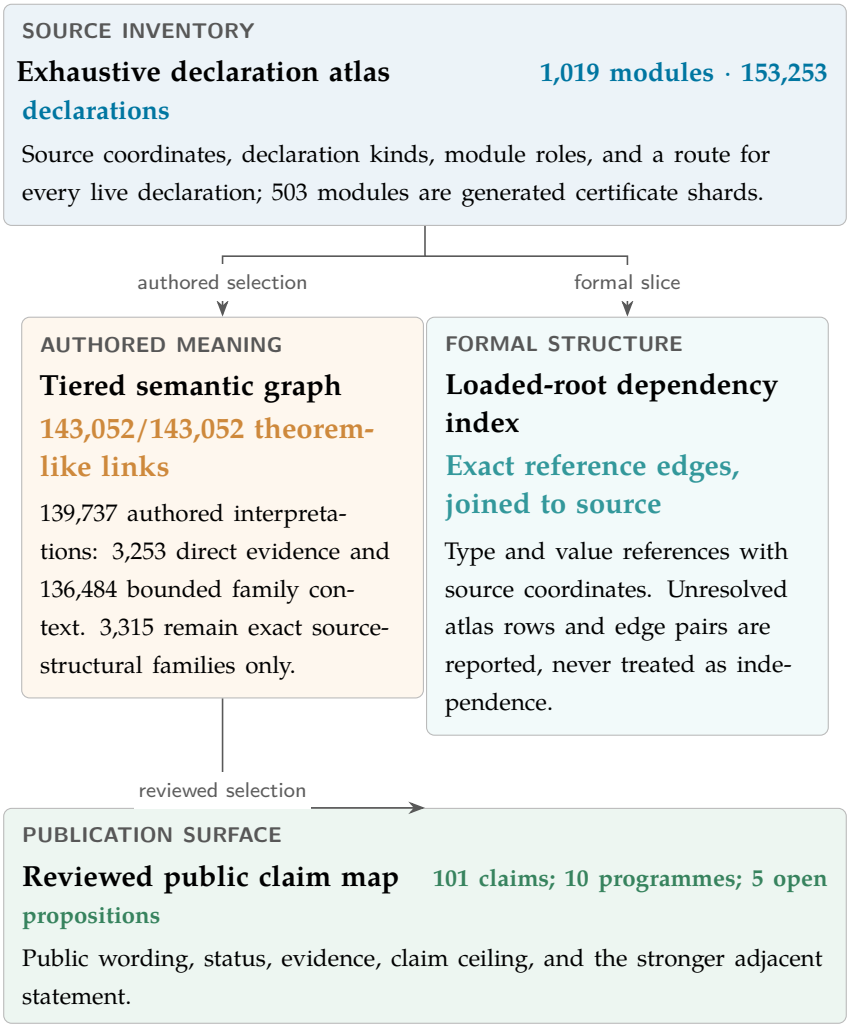

Figure 1: The navigation layers. The dependency index and semantic graph are different partial views of the exhaustive atlas; neither is presented as a complete interpretation.

Keeping the tiers visible prevents exhaustive linkage or bulk helper assignment from being misreported as exhaustive direct understanding. A paper-seeded population query continues to rank exact live citations whose best route is structural rather than authored.

Reviewed public claims. The claim map selects 101 results for public exposition. It records 37 as proved here, 8 as formalised here, 5 as unconditional progress, 39 as conditional reductions, 7 as verified finite instances, 3 as cited only, and 2 as open. Five explicit frontier propositions describe the stronger obligations that survive. These labels are maintainer-reviewed public meaning, not outputs inferred by the proof kernel.

3 A bounded tour over an unbounded drilldown

The first command returns a six-line card rather than a database dump. It derives corpus scale, formal-graph scale and misses, the authority boundary, the eight-problem

map, the open frontier, and the available intent classes from the committed projections. The full packet uses a registry-scaled budget: 18 kB of base context plus 2 kB per indexed problem, hence 34 kB for the present eight-problem registry. It expands the card into a mathematical map, status counts, reader-specific contracts, and typed follow-up commands.

Five intent lenses cover the main transitions:

1. understand the mathematics through a selected programme and its claims;
2. locate any formal object through ordinary-language search followed by exhaustive inventory;
3. inspect exact formal dependencies through proof cones and paths;
4. begin a checked change through the session notary and focused builder;
5. audit the agent and release system through the publication architecture and release gate.

The route logic names no specimen theorem or module. Repository-specific mathematics enters through data: programme rows, claims, declaration records, and frontier propositions. This distinction matters. The mechanism is generic over the repository's projection schemas, while the current projection content is intentionally specific to the mathematics it describes.

Ordinary-language entry is likewise not an undocumented trigger list. The `--vocabulary` packet is an executable Rosetta surface: it publishes question operators, aliases, transparent expansions, typed route hints, an all-problem registry projected from `docs/problems.json`, and a route-discovery lexicon projected from `docs/claims.json::machine_readable_paper.entrypoints`. A unique exact authored term may take a constant-time fast path; ambiguous or unseen language falls through to visible vocabulary interpretation and ranked corpus search. Every search response records that interpretation, so an agent can inspect how its wording was translated instead of depending on phrase luck.

The packet also answers four different cold readers. A research mathematician gets programmes, claim statuses, and surviving open propositions. A formalisation engineer gets declaration coordinates, exact value references, and a route into a checked change. An AI-lab researcher gets coverage and authority distinctions. An independent contributor gets the no-build orientation, changed-cone build route, and public handoff check. Each reader sees the same facts through a different question contract rather than a separate hard-coded example.

4 Crossing from navigation to authority

The architecture treats navigation and proof as separate capabilities (Figure 2). A static query may nominate a theorem or dependency path. It cannot establish that a proposed application typechecks, that a proof closes a goal, or that an English interpretation is mathematically faithful.

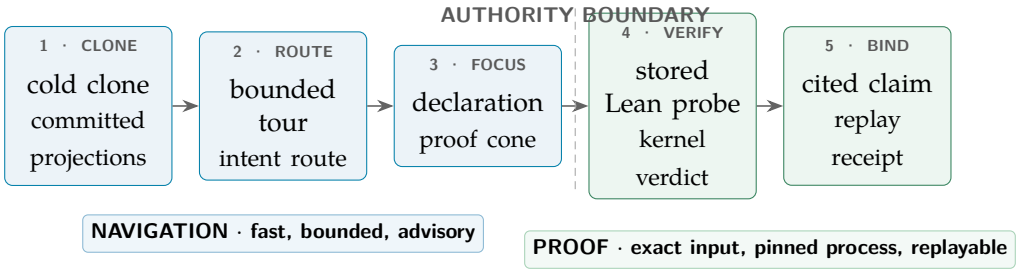

Figure 2: The shortest path from first contact to an authoritative receipt. The agent chooses the route; the pinned Lean process supplies the verdict.

The session notary records a typed move grammar in an append-only ledger: observations, conjectures, plans, interpretations, abandonments, probes, claims, and closure. Conjectures may declare a falsifier. Dead ends remain in the record instead of being rewritten into a linear success story.

A probe is different. The notary stores the exact Lean input bytes, runs the pinned Lean process, and computes one of three verdicts from the process result: accepted, accepted with an admitted placeholder, or rejected. The agent cannot type a verdict into the ledger. A claim must cite an accepted probe receipt; the notary rejects weaker citations. Replay reruns every stored probe and compares the current result with the recorded verdict.

This design deliberately leaves search policy open. LeanDojo couples an open programmatic Lean environment with extracted proof data, premise annotations, retrieval, and a theorem-proving benchmark [4]. Pantograph exposes tactic execution, proof states, metavariables, and data extraction through machine interfaces [5]. Those systems make machine-to-Lean interaction richer. The notary here addresses a complementary question: after an agent chooses its own policy, which parts of the resulting reasoning record are advisory, and which exact claims are grounded by replayable kernel acceptance?

5 Compilation after comprehension

A committed navigation projection is portable text; an `.olean` file is a toolchain- and platform-specific build product. A cold clone can therefore orient immediately but must either compile the selected formal target or restore a compatible cache before editing.

The focused build wrapper accepts an exact target, modules changed since a Git reference, or the stale cone derived from Lake traces. It orders the selected modules into dependency waves, limits parallel jobs, and finishes with a serialized Lake authority check. Restored caches use Lake's content traces rather than checkout timestamps. In one historical four-file addition, the planner selected 26 modules in four waves rather than both complete roots; in a warm two-root dogfood plan it selected no stale modules. The test suite covers trace freshness, changed-file selection, dependency ordering, bounded parallelism, failure propagation, and the final authority check.

The exhaustive dependency-index validator is expensive for a different reason: it traverses the elaborated environment and exports direct references. After a full export

matches the committed index, the validator writes an exact receipt below the ignored Lake cache. Its key includes every supported Lean source file, the toolchain and Lake locks, the exporter and builder inputs, the declaration atlas and generated manifest, the formal-source release slice of the claim registry, the dependency-extraction helper definitions, and the committed index bytes. An ordinary check reuses the receipt only while all inputs and output remain byte-identical; an explicit full-check mode bypasses it. A changed formal or exporter input therefore forces a fresh incremental build and full export, while documentation-only revisions and exact cache restores do not.

This is intentionally conservative. The receipt does not claim that the environment export itself has become incremental. It prevents unnecessary repetition when its entire input surface is unchanged. When Lean source changes, Lake still recompiles only stale dependencies, but the graph exporter currently scans the supported environment again.

6 Dogfood receipt

The committed prospective session is a naturalistic use of the workbench, not a synthetic benchmark. The ledger contains six reasoning notes, one kernel-accepted probe, and two claims bound to that probe. It produced a binary carry-pivot normal form for numeral-adjacent boundary words and an exact divisor-incidence identity that sharpens a previously landed one-sided coefficient bound to equality. At the audited revision, replay reruns the stored probe and returns the same accepted verdict.

The receipt supports a narrow statement: the stored input is accepted by the pinned Lean environment, the cited claims point to that accepted probe, and the result survives replay at this revision. It does not establish that the agent's route was optimal, that every candidate was searched, that a bounded failure is impossible, or that the mathematics is new outside the repository history.

The navigation surface was dogfooded separately as four audiences. The research-mathematics route reaches the programme map and explicit frontier; the formalisation route reaches a source coordinate and exact proof cone; the AI-lab route exposes exhaustive-versus-selective coverage and receipt authority; the contributor route reaches a focused build and release check. Adversarial tests remove or distort each protected transition and require the cold-clone validator to reject the mutation. This evaluates the declared first-contact contract, not subjective comprehension.

7 Relation to prior work

Proof blueprints pair informal plans with named Lean declarations and author-supplied dependency links. `leanblueprint` stores a blueprint in TEX and its checker only confirms that each named declaration exists [1]; it neither infers those links nor checks that the informal and formal statements agree. LeanArchitect infers formal dependencies and proof-hole status and exports synchronised blueprint material [2]. The Carleson project shows a blueprint used as a large collaboration surface. Its public blueprint split the formalisation into about 180 claimable tasks, most corresponding to one blueprint lemma; formalisation feedback produced localized corrections, modifications, and extensions to the mathematical guide, including a few changes to the general setup and

main theorems [3]. The present tour has a different first-contact role. It begins after the repository already contains a large corpus, distinguishes exhaustive inventory from authored interpretation, and routes a reader into exact dependency and proof-receipt tools.

LeanDojo and Pantograph provide stronger interaction substrates for learned or scripted theorem proving [4, 5]. This work does not propose a new proof-search policy. It makes the policy slot explicit and records enough of an agent-chosen trajectory to separate notes, nominations, rejected probes, and accepted claims.

Nor is its declaration graph or retrieval route novel in isolation. LeanGraph extracts typed elaborator-level edges across Lean projects, while a separate network study analyses Mathlib's multilayer declaration graph [6, 7]. LeanExplore combines semantic, lexical, and graph ranking behind Python and MCP interfaces, and LeanSearch v2 targets global premise sets through iterative sketch–retrieve–reflect [8, 9]. The narrower claim here is that one cold-clone contract composes source addresses, selective semantics, dependency cones, incremental validation, and authority receipts without requiring a hosted service or an initial Lean build.

Large-library maintenance supplies the relevant compilation lesson. *Growing Mathlib* describes performance-aware library design, deprecation, semantic linters, benchmarks, review tooling, and explicit technical-debt management [10]. The changed-cone planner and receipt cache apply a smaller-scale version of that maintenance posture. The local measurements do not transfer Mathlib's scale or social evidence.

Lean Atlas narrows the declarations whose meaning can affect selected theorem statements and leaves semantic verification to people [11]. That is close to the authority distinction here: formal dependency can focus human inspection without deciding intended meaning. The cold-clone tour adds a front-door contract across exhaustive inventory, selective semantics, public claims, and agent receipts.

8 Limits and transfer conditions

This is a case study of one repository at one audited revision. Its projection schemas and routing engine are reusable mechanisms, but another project must supply its own source roots, programme map, claim vocabulary, frontier statements, and authority owners. The paper gives no evidence that a new project can adopt the architecture cheaply or that agents using it outperform agents with an ordinary README.

Counts describe artifact coverage, not mathematical understanding. Direct dependency edges are not proof explanations. Authored semantic nodes may be wrong or incomplete. Maintainer-reviewed claims may also be wrong; automation preserves recorded relationships but does not judge unrestricted prose.

Receipted reasoning is auditable rather than necessarily good. A model may choose unproductive probes, omit a relevant source, or rationalise a dead end. Replay detects environment drift in stored probes but not every flaw in the surrounding interpretation. Context-blind historical experiments also cannot establish training-data blindness.

Finally, the first full project build and the first elaborated graph export remain material costs. The architecture moves them after comprehension, supports compatible

cache restoration, and avoids repeating an exact export when nothing relevant changed. It does not eliminate the cost of validating a genuinely changed formal environment.

9 Authority-preserving agent entry

An agent-native formal repository should not merely contain many proofs. It should reveal, from a cold clone, what mathematics exists, what each navigation layer knows, what remains open, and which next action crosses into proof authority. The architecture here realises that sequence with committed projections, a bounded intent tour, exact loaded-root dependency data, a session notary, focused builds, and cache-bound validation receipts.

The central design choice is separation. Comprehension comes before compilation; nomination comes before application; reasoning notes remain distinct from kernel verdicts; exhaustive routing remains distinct from selective interpretation; and a public claim remains distinct from the formal statement it describes. Those boundaries let a capable agent move quickly without making speed look like authority.

A Reproduction routes

The public repository commits the tour, route, projections, workbench session, and validators used in this paper. The shortest reproduction route has three bounded tasks.

1. Orient without elaborating Lean. These commands inspect committed projections only:

```
python3 scripts/query_corpus.py --tour --format card
python3 scripts/query_corpus.py --route agent_native_corpus_navigation
python3 scripts/query_corpus.py --search <ordinary-language-query>
python3 scripts/query_semantic.py inventory <candidate-name> --limit 1
python3 scripts/check_cold_clone_comprehension.py --quick
```

The full tour packet is obtained by omitting the format flag. A declaration name returned by inventory can be expanded into a direct neighbourhood or bounded proof cone.

2. Cross into proof authority. Beginning formal work enters the notary and then the focused builder:

```
python3 scripts/proof_workbench.py open --help
python3 scripts/lean_fast_build.py --jobs 2 --lake-staleness <target>
```

3. Audit dependency-index reuse. The ordinary check may reuse an exact cache receipt; the full check bypasses it:

```
python3 scripts/test_lean_dependency_index_cache.py
python3 scripts/build_lean_dependency_index.py --check
python3 scripts/build_lean_dependency_index.py --check --full-check
```

The first ordinary check after a cold clone may perform the full export; a matching restored Lake cache can carry the exact receipt. The full-check form always bypasses it.

References

[1] P. Massot, *leanblueprint*, plasTeX plugin for Lean formalisation blueprints, 2020, software repository, accessed 28 July 2026.

[2] T. Zhu, P. Monticone, S. Welleck, and J. Avigad, *LeanArchitect: Automating Blueprint Generation for Humans and AI*, in *17th International Conference on Interactive Theorem Proving*, LIPIcs 382, 2026, pp. 25:1–25:16, DOI.

[3] L. Becker et al., *A Blueprint for the Formalization of Carleson's Theorem on Convergence of Fourier Series*, 2025, arXiv:2405.06423.

[4] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar, *LeanDojo: Theorem Proving with Retrieval-Augmented Language Models*, in *Advances in Neural Information Processing Systems 36*, 2023, arXiv:2306.15626.

[5] L. Aniva, C. Sun, B. Miranda, C. Barrett, and S. Koyejo, *Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem Proving, High Level Reasoning, and Data Extraction in Lean 4*, in *Tools and Algorithms for the Construction and Analysis of Systems*, 2025, pp. 116–137, DOI.

[6] S. Kurgan et al., *TheoremGraph: Bridging Formal and Informal Mathematics*, 2026, arXiv:2606.25363.

[7] X. Li, N. Peng, S. Severini, and P. Shafto, *The Network Structure of Mathlib*, 2026, arXiv:2604.24797.

[8] J. Asher, *LeanExplore: A Search Engine for Lean 4 Declarations*, 2025, arXiv:2506.11085.

[9] G. Gao et al., *LeanSearch v2: Global Premise Retrieval for Lean 4 Theorem Proving*, 2026, arXiv:2605.13137.

[10] A. Baanen, M. R. Ballard, J. Commelin, B. Ginge Chen, M. Rothgang, and D. Testa, *Growing Mathlib: Maintenance of a Large Scale Mathematical Library*, in *Intelligent Computer Mathematics*, 2025, arXiv:2508.21593.

[11] B. Yanahama and A. Sannai, *Lean Atlas: An Integrated Proof Environment for Scalable Human–AI Collaborative Formalization*, 2026, arXiv:2604.16347.