From Spare Compute to Cumulative Mathematics

An open-source strategy for agent-assisted research on hard problems

Will Cook*

August 2026

Abstract

This paper describes an open-source strategy for agent-assisted mathematical research. The starting point is a practical asymmetry. Building a durable research environment requires navigation, experiment records, formal proof checking, claim boundaries, review tools, and attribution machinery. Once that fixed cost has been paid, another person should not have to rebuild it in order to contribute a mathematical idea or a night of spare compute.

The proposed research commons accepts four independently useful inputs: compute, mathematical direction, infrastructure improvements, and review. Its outputs are not limited to solved problems. Proofs, counterexamples, finite computations, corrected statements, failed mechanisms, formal no-go theorems, and better research tools all reduce the cost or improve the direction of later work. Accepted mathematical returns enlarge a versioned problem corpus; accepted architectural returns improve the process that turns future compute and insight into reviewable mathematics. Provenance and role-specific credit remain attached when a contribution is assimilated.

The implemented case is a public Lean repository organised around eight open Erdős problems. The problems were chosen as difficult research worlds, not because the present system was expected to solve them. A fresh clone contains papers, formal source, explicit open boundaries, corpus queries, validation programs, and structured return paths. All eight problems remain open. The repository is therefore a proof of concept for cumulative research infrastructure, not evidence that distributed agents already outperform mathematicians or that additional compute will solve a named problem. As of 31 August 2026, the author had not recorded a completed external cold-clone use or an accepted external contribution, so the contributor experience remains untested outside author-operated runs.

The paper gives a production model, contribution protocol, trust and security boundary, attribution policy, comparison with volunteer computing, Polymath-style collaboration, formalisation projects, and multi-agent proof systems, and an evaluation agenda. Mathematics is the initial domain because its objects are digital and formal claims can be checked incrementally. Any later extension to experimental science would require domain scientists, physical laboratories, safety governance, and new evidential authorities.

What is implemented

Contribution. A public contribution protocol accepts compute, mathematical direction, infrastructure, and review as independently useful inputs, while preserving evidence boundaries and role-specific credit. **Implementation.** One fresh clone contains eight problem worlds, formal source, explicit open obligations, validation programs, and structured return paths. **Limit.** All eight problems remain open, and no external contribution has yet tested the proposed contributor experience.

Authorship and AI use. Every word of this manuscript was generated by agents based on large language models operating within Will Cook's private research system for artificial intelligence. 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. This production disclosure supplies no independent verification; Cook did not independently verify every mathematical claim.

1 The strategy

This project began with one undergraduate working without an institutional research team or a dedicated compute allocation. That fact does not validate the mathematics or the architecture. It makes the strategic question concrete: can the fixed cost of research infrastructure be paid once, published, and then shared by people who bring different scarce inputs?

The intended answer is an open research commons. A mathematician may supply a theorem, counterexample, reference, or correction. A Lean contributor may formalise or audit one statement. A person with spare compute may run an agent against a bounded public frontier. An infrastructure contributor may improve navigation, experiments, validation, reproducibility, or the public contribution path. A reviewer may reconcile a formal statement with its intended meaning or decide that a returned result does not survive scrutiny. These are different contributions. None should have to masquerade as a solution to receive credit.

The project is AI-native in a narrow, declared sense. Contributors may use language models and agent harnesses throughout the work, provided that their use is disclosed. Model assistance is neither a defect nor evidence of mathematical value. What enters the accepted record is the attributable delta: a mathematical idea, a question or direction that produces useful mathematics, a proof or counterexample, a review, or an architectural change that makes later research more effective. Each still needs the evidence and review appropriate to its claim.

The repository is deliberately centred on hard problems that current systems cannot simply dispatch in one prompt. The purpose is to expose the complete research process: selecting a useful subproblem, reading prior work, testing conjectures, recording failed routes, formalising stable steps, explaining the result, and preserving the exact statement that remains open. A solution is one possible output of this process. The cumulative record is the output that can be produced on every serious run.

Each problem is therefore treated as a continuous goal rather than a sequence of blank-slate prompts. The endpoint persists while the frontier changes. A new agent run starts from the accumulated theorems, computations, counterexamples, failed mechanisms, citations, and open obligations; chooses a direct mathematical transition worth attempting; and returns an inspectable delta plus the boundary that survived. It may hand back a theorem, a narrower reduction, a killed route, a source correction, or a better next question. When the endpoint remains open, that is not a failed run. Losing the state and paying to rediscover it would be.

Two coupled goals, not one endless agent

The most useful public deployment has two persistent roles. A *discovery goal* stays close to one mathematical frontier: it reads the corpus, compares attacks, uses computation to discriminate them, attempts ordinary and formal proofs, and returns the smallest stable theorem, counterexample, no-go, computation, or corrected boundary. A *stewardship goal* stays close to the corpus as a whole: it compares the new object with existing results, decides which declarations form one mathematical family, reconciles the paper and assurance surfaces, and returns the strongest next question to the discovery goal. The corresponding systems contract is stated in the coupled-goals section of the systems paper.

The two roles can be run by different people, agent harnesses, or machines. A compute contributor may operate only the discovery side. A mathematician may supply a direction, assess whether a result is genuinely nontrivial, or repair the statement and exposition without paying for a long model run. An expositor may improve the paper projection; a Lean contributor may strengthen the formal interface; an agent builder may make the loop cheaper. All of these acts remain attributable contributions to one evolving mathematical record.

The stewardship role is not a ceremonial reviewer added after the work. Its appraisal changes where future effort goes. It may discover that a new "theorem" is only a reformulation, that five declarations are one coherent result family, that a short no-go eliminates an expensive research direction, or that a paper still leads with a weaker theorem. It then updates four separate outputs: proof and evidence status, mathematical appraisal, paper prominence, and the next allocation of compute or expert attention. These outputs may influence one another, but they are not one score and none can promote an unproved claim.

The coupling is event-driven. A stable mathematical delta wakes the steward; a changed appraisal, missing consumer, or sharper open boundary can wake the miner. An unchanged repository should

consume no agent turn merely to report that it is unchanged. This makes continuous work a sequence of inspectable state transitions rather than an expensive synonym for leaving a chat window open.

The public `run-coupled-research-goals skill` makes this control shape executable in a cold clone. It invokes the existing mining and consequence-propagation jobs, preserves a shared source pin, and passes committed objects and receipts rather than conversational claims of progress. One agent may alternate between the roles, or different people, models, subscriptions, and machines may supply them. The architecture requires distinct decisions, not an unnecessarily grand collection of laptops.

This is not a proposal to place a public queue in front of a private machine. The public clone must stand on its own. The larger private workbench explains how the initial corpus was produced, but it grants no proof or publication authority to a returned result. Contributors may use any model runner or no model at all. What joins their work is the public problem definition, evidence contract, and review path.

The systems paper gives the complete claim-transition lifecycle. The cold-clone paper tests how a new agent reaches the public mathematics and the validation boundary. This paper owns the participation, credit, and growth strategy; the three papers therefore describe different parts of one prototype rather than competing accounts of it.

What one clone lets a contributor do

A contributor does not need to understand the whole repository before doing useful work. From one clone, a person or agent can choose one of five first actions: mine a bounded problem route; contribute mathematical direction without paying for the compute that follows it; formalise or review one claim; repair the research machinery; or propose another sourced problem world. The clone-local `explain-public-system skill` lets an agent read the public corpus and companion papers on the newcomer's behalf, explain the claim boundaries at the requested level, and point back to exact evidence. The reader can therefore begin with the ordinary request "explain this repository to me" rather than first mastering its file layout. The other skills select and run a frontier, coordinate the coupled goals, install the same workflows in a compatible agent harness, and describe the present multi-stage process for adding a problem.

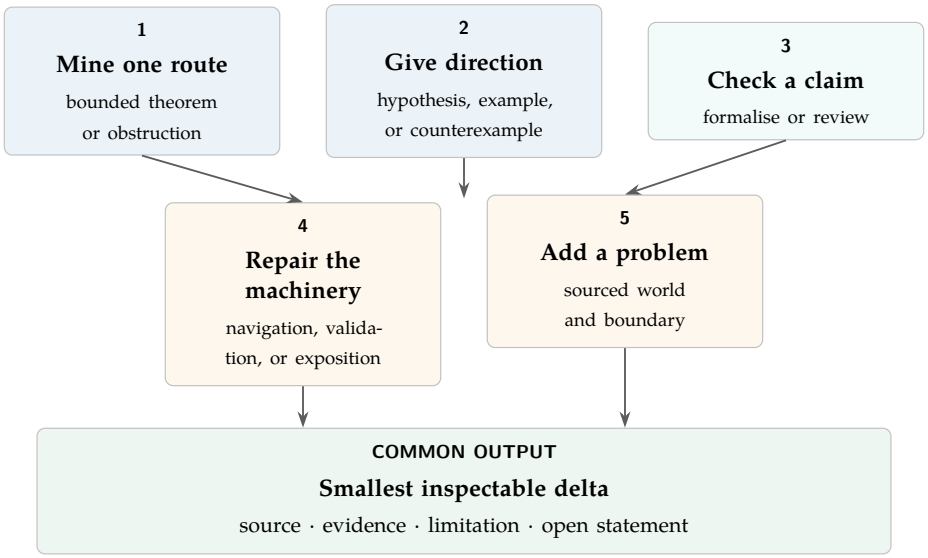

Figure 1: Five useful first actions. None requires an endpoint solution, and none receives a stronger status than the evidence returned with it.

2 A production model for mathematical progress

Candidate production depends jointly on compute, model capability, human direction, infrastructure, problem choice, and accumulated knowledge. Mathematical progress additionally depends on validation and scarce human review. The notation below makes those dependencies explicit. Let

$$C_t,\ A_t,\ H_t,\ I_t,\ P,\ K_t,\ V_t,\ R_t$$

denote, at time t, available compute, agent capability, contributed human mathematical direction, infrastructure quality, problem selection, accumulated research knowledge, validation capacity, and review capacity. The rate of candidate generation may be written schematically as

$$G_t = G(C_t, A_t, H_t, I_t, P, K_t). \tag{2.1}$$

The rate of *reviewable mathematical progress* is a different quantity:

$$Y_t = Q(G_t, V_t, R_t), \tag{2.2}$$

where Q includes formal checking, statement reconciliation, attribution, and editorial selection. Equations (2.1) and (2.2) are a conceptual production model, not a fitted empirical law. They record complementarity and bottlenecks. More compute can generate more attempts without increasing the rate at which strong results are validated or understood. A better model can still revisit a closed route if the corpus is difficult to navigate. Expert insight can change the search distribution without supplying any compute. Review capacity can be the binding constraint even when proof generation is cheap.

The open-source strategy adds two recursive updates:

$$\begin{aligned} K_{t+1} &= K_t + \Delta_t^{\text{theorem}} + \Delta_t^{\text{counterexample}} + \Delta_t^{\text{no-go}} + \Delta_t^{\text{computation}} + \Delta_t^{\text{correction}}, \\ I_{t+1} &= I_t + \Delta_t^{\text{navigation}} + \Delta_t^{\text{validation}} + \Delta_t^{\text{reproducibility}} + \Delta_t^{\text{governance}}. \end{aligned} \tag{2.3}$$

Only reviewed, scoped returns enter these updates. The first line makes later research less forgetful. The second makes later research less expensive or more reliable. A local failure can therefore have two legitimate effects: it may add a mathematical obstruction to K_t, or reveal a reusable process defect whose repair improves I_t. It cannot change a conjecture's truth status merely because it changed the process.

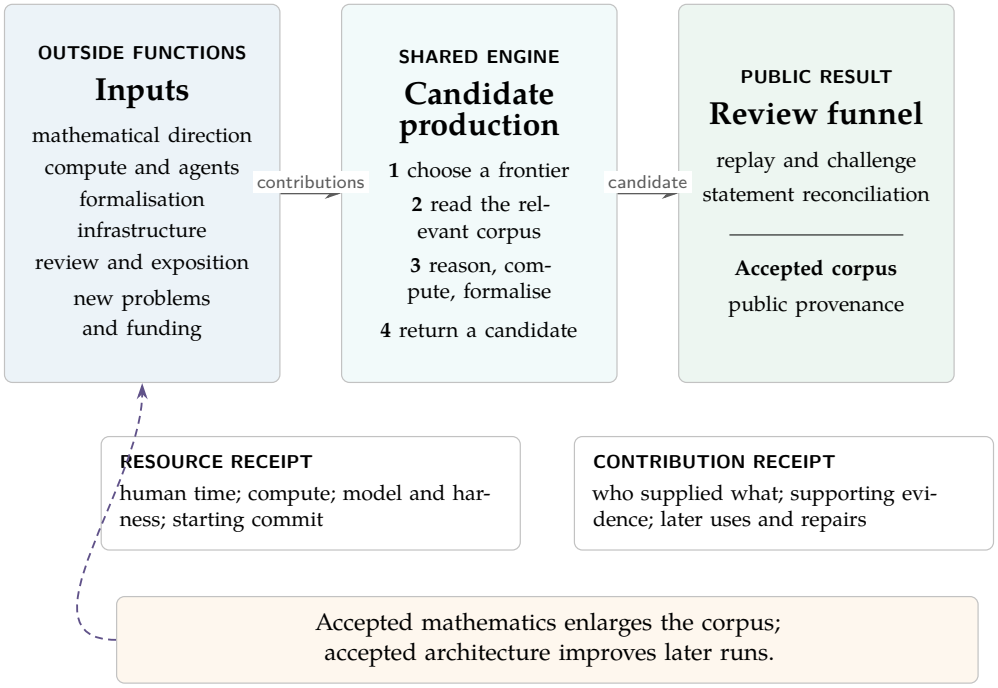

Figure 2: The conversion loop. The labels on the left are functions, not classes of people: one person may perform several, and several people may perform one. Search output crosses replay and review before it enters the accepted corpus. Mathematical returns enlarge the corpus; architecture returns alter the engine.

The model also gives a longitudinal use for the repository. A fixed, versioned problem world can be revisited as models, runners, and compute budgets change. Comparisons must disclose the starting commit, model, scaffolding, number of attempts, compute, and review procedure. Otherwise a success says little about which variable changed. A public failure record is equally important: capability claims are badly distorted when successes are announced and the number and cost of failed attempts are hidden [1].

3 Why begin with hard mathematics

Mathematics is an unusually clean first domain for this strategy. The objects, programs, papers, and formal statements can all travel in one clone. Many conjectures can be probed by exact computation. A proof assistant can check a stable formal step without waiting for an entire long argument to be written informally and retrofitted later. This does not make mathematical meaning automatic: Lean checks the proposition in the source, not whether it is the proposition the project intended to study.

Computation has a second role. It can provide an artificial form of local intuition to a system that is better at writing and running code than at reproducing a mathematician's tacit judgement. Exact experiments can expose counterexamples, compare representations, and locate a narrow regularity. Their result updates route selection and conjecture pressure. It does not change the evidence class of an unbounded statement. In Bayesian language, an experiment can update which route deserves attention; it cannot assign a formal posterior probability to a theorem or replace proof.

The eight current problems were selected as demanding research environments, not as a claim that they were the eight most tractable or valuable open problems. Their papers expose materially different contribution points.

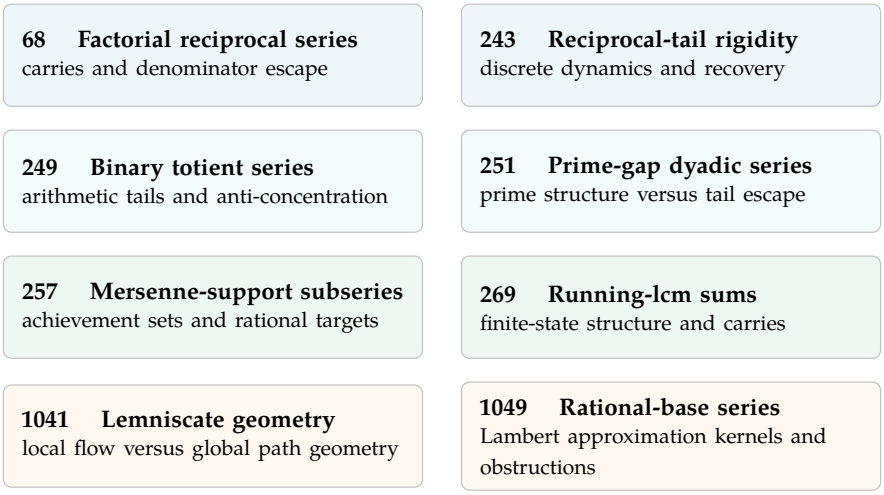

Figure 3: Eight distinct research environments. The results map and the problem papers state their exact frontiers; this strategy paper does not reproduce the specialist vocabulary needed to attack them.

Every row admits more than theorem proving. A mathematician may sharpen a hypothesis or supply a counterexample. A programmer may run a discriminating exact calculation. A formaliser may turn a paper deduction into a checked declaration or find that the two statements disagree. A literature reader may locate a theorem that closes or invalidates a route. The contribution is the exact returned object and its limitation, not the prestige of the problem number.

4 The public research object

The unit of participation is a *problem world*. A problem world includes the endpoint question, current public status, principal theorems, formal source, experiments, known counterexamples, failed mechanisms, source literature, and exact open obligations. A bounded task is a neighbourhood in that world, not an invitation to read the repository indiscriminately.

The public layers carry different kinds of evidence. Their order is easier to read as an authority ladder than as one all-purpose badge.

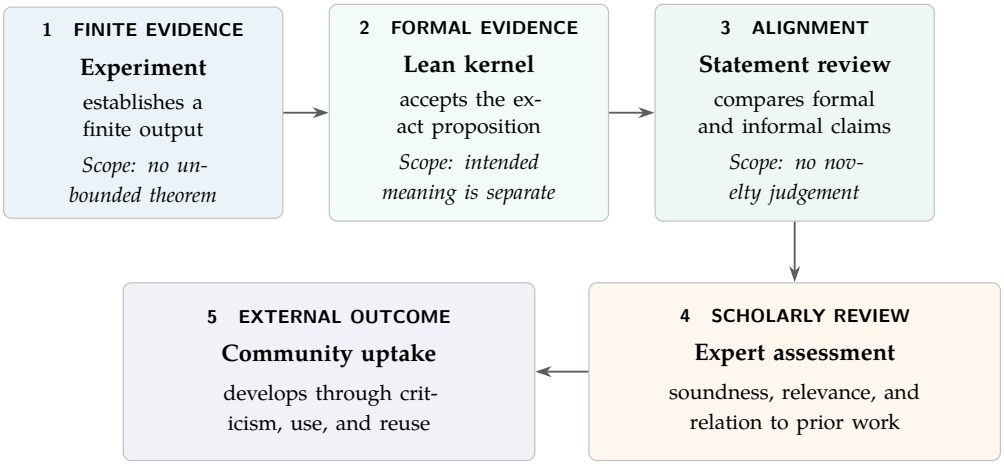

Figure 4: Evidence and acceptance remain separate. A candidate need not pass through every box in a single line, but no earlier box inherits the authority of a later one.

The project is a composition layer around substantial prior work, not a replacement for it. Erdős Problems supplies questions, sources, status, and a problem community [7]. Lean supplies the formal language and kernel [8]. DeepMind's `formal-conjectures` supplies a public one-problem-file model with metadata, snapshots, issues, and pull requests [9]. Comparator and Palomar supply exact-interface checking, permitted-axiom inspection, and a durable formal record with a deliberately limited editorial claim [10]. Polymath supplies a social precedent for publishing small, tentative, and negative contributions [4]. BOINC and GIMPS supply the volunteer-compute precedent and make visible why executable work, independent checks, and legible credit matter [2, 3]. The present repository wraps these practices into a route that a newcomer can enter without first rebuilding each component.

The same underlying substrate can support several reader projections: a short public primer, a specialist paper, a detailed proof account, Lean declarations, an agent explanation, and machine-readable queries. Each projection must link back to its source claims and evidence. A shorter or friendlier account does not acquire permission to strengthen them.

An external runner begins at `the compact agent entry`. It can inspect all problem frontiers, choose a bounded question, read the relevant paper and source neighbourhood, run experiments or edit formal code, validate the result, and prepare a return. None of these steps requires access to the private workbench. A contributor is free to replace the agent, the scheduler, or the entire search policy while retaining the public evidence boundary.

5 The contribution protocol

The ordinary Git verb is a pull request. A contributor forks or clones the repository, creates a branch, commits a focused change, pushes the branch to a public fork, and opens a pull request. A person who has a result but no patch can open a structured research-progress issue. A person with an infrastructure proposal can use the parallel architecture-proposal path.

The clone-local `submit-pull-request` skill makes this sequence agent-readable. It separates mathematically or architecturally coherent changes into reviewable commits when they can genuinely be separated, runs the narrow checks for each change, and prepares the repository's pull-request template. Commit boundaries follow the meaning of a change rather than a rule of one file or one diff type per commit. Pushing to a fork and opening the pull request are external actions and occur only after the contributor authorises them.

Upstream work may continue while the contributor's branch remains open. The recorded starting commit supplies the common ancestor from which Git recovers the original delta; the contributor does not need a copy of the maintainer's private or current working tree. Review therefore has two passes. First, reproduce the change in its original public context. Second, reconcile the accepted substance with

current main and rerun the present checks. A material conflict resolution is a new, separately credited integration change. It does not erase or retrospectively rewrite the original contribution.

The complete protocol has nine steps.

1. **Clone.** Record the exact public commit and environment.
2. **Orient.** Read the claim boundary and the paper governing the selected problem or architecture area.
3. **Select.** Name one bounded frontier, expected evidence, and a stop condition. Working on all problems is allowed, but each returned result remains independently scoped.
4. **Work.** Use any mixture of reasoning, source research, computation, Lean, and human mathematical judgement. Preserve useful negative results instead of rewriting the run as a linear success.
5. **Validate.** Run the checker appropriate to the claim. A computation receives a computation receipt; a formal theorem receives a Lean receipt; neither receives semantic or publication authority from that fact alone.
6. **Propagate.** Enumerate the plausible Lean, claim, paper, computation, route, validation, and contributor consumers. Update each one, verify it unchanged, defer it with a re-entry condition, or explain why it is not a consequence. Repeat this pass after old-branch work is reconciled with current main.
7. **Return.** Submit source, evidence, starting commit, tool disclosure, collaborators, requested contribution roles, and the exact stronger statement that remains unproved.
8. **Review and adopt.** Reviewers reproduce the evidence, reconcile formal and informal statements, decide scope, and either accept, request revision, reject, or preserve the return as a scoped negative result.
9. **Publish and credit.** An accepted generation is linked to the paths and commit that assimilated it. Later corrections append lineage; they do not erase the earlier contributor.

The accessibility claim is role-bounded. A non-specialist compute contributor is not asked to decide what constitutes a proof or to invent a mathematically meaningful objective unaided. The default work unit is a packet prepared from an authored mathematical route: an exact open statement, a relevant source neighbourhood, a permitted experiment or proof obligation, an expected evidence class, and a stopping condition. Qualified mathematical review should precede large-scale distribution; it is not claimed for every present route. The contributor and agent execute the packet and return evidence. Lean checks a formal proposition. A qualified reviewer checks whether that proposition says what was intended and whether the return is relevant. The return remains a candidate until the applicable gates have been crossed.

The mathematician who supplies an input and the mathematician who verifies a return are also distinct roles. The first may contribute a conjectural mechanism, a counterexample pattern, or simply a well-posed question that causes the system to produce useful mathematics. The second inspects the resulting statement, argument, prior art, and significance. One person may perform both roles, but the provenance record must not let an input certify its own output merely because it came from a mathematician.

This protocol supports two first-class tracks. The *mathematics track* accepts proofs, reductions, computations, counterexamples, corrections, formalisation, and reproducible failed routes. The *architecture track* accepts improvements to agent workflow, navigation, validation, reproducibility, public experience, governance, and tooling. An architecture return never needs a fictitious problem number, and it cannot request promotion of a mathematical claim.

The distinction between a return and an adopted result is load-bearing. A pull request is a proposed delta. A review act records a decision. A release generation is an immutable public state. Keeping these separate permits credit for a useful counterexample, correction, or design even when no theorem is merged.

6 From an agent claim to mathematical attention

An agent saying "solution found" is an event in a run log, not a project claim. An informal derivation may contain an unnoticed gap. A Lean theorem may be completely kernel-correct while formalising the wrong statement, assuming an inadmissible axiom, or proving a result too weak to settle the problem. Formal verification therefore strengthens one layer of evidence; it does not collapse correctness, intended meaning, novelty, significance, and community acceptance into a single status.

The project uses a deliberately asymmetric attention funnel:

1. **Candidate.** A person or agent returns a scoped argument, computation, formal declaration, counterexample, or no-go.
2. **Replay.** Automated checks reproduce the code and attach the correct evidence class without promoting its claim.
3. **Internal challenge.** Independent agents or contributors try to break the statement, locate stronger prior art, vary the experiment, and reconcile formal and informal formulations.
4. **Maintainer triage.** A human checks whether the bundle is coherent enough, relevant enough, and sufficiently de-risked to justify scarce specialist attention.
5. **External record.** A mature Lean result may be packaged through Comparator and submitted to Palomar; a mathematical account relevant to a listed problem may be placed before the Erdős Problems community and the appropriate research audience.
6. **Acceptance.** Mathematicians inspect, explain, criticise, reuse, publish, or reject the work over time. Broad mathematical acceptance is exogenous to this repository and cannot be granted by its maintainer.

Palomar is valuable precisely because its claim is narrower than peer review. It mechanically verifies a pinned challenge–solution interface using Comparator and kernel checking, records structured disclosure, and applies a documented automated editorial filter. It explicitly does not endorse the result, establish novelty, or provide human expert review [10]. Likewise, the Erdős Problems forum asks long proofs or partial proofs to be linked as external documents and filters implausibly incomplete claims; it is a route to relevant attention, not a substitute for that community's judgement [7].

Multiple contributors can improve triage, but headcount is not mathematical evidence. What raises a bundle's priority is costly, legible corroboration: independent replay, adversarial review, distinct contributors repairing different failure modes, a stable formal interface, and an exposition that a specialist can audit. This social proof should allocate attention, never inflate the theorem status. It lets a contributor without institutional connections accumulate a public chain of reasons for an expert to look, without requiring the expert to sift every raw agent transcript.

7 Progress, provenance, and credit

The project recognises progress in evidence classes rather than by counting commits or theorems. A return may contain:

- an unconditional theorem or a formal proof of an existing statement;
- a counterexample or corrected hypothesis;
- a no-go theorem excluding a defined strategy class;
- a bounded computation with code, range, and interpretation;
- a conditional reduction with its antecedent still visible;
- a reproducible failed route that prevents duplicated search;

- a literature correction or exact source locator;
- an architecture proposal or validated infrastructure change; or
- exposition or review that changes what another person can understand and check.

These items should not be placed on one scalar leaderboard. Compute volume, mathematical depth, software quality, correction work, exposition, and review are not commensurable. A public recognition view can instead expose the contributor, accepted artefacts, evidence classes, problem or architecture area, and contribution roles. The current receipt schema supports the fourteen CRediT roles as an optional vocabulary, including conceptualisation, methodology, software, validation, investigation, and writing [13]. CRediT describes work; it does not decide authorship.

The default promise is simple: the project does not demand ownership of a contributor's underlying idea in exchange for adoption. If a mathematical or architectural contribution is assimilated, the public lineage should continue to name who originated it and what survived. Substantial authorship and formal publication decisions remain separate judgements. A role ledger is more honest than either a winner-takes-all solver credit or a pseudonymous collective record when the project can preserve exact provenance.

The same rule applies when a local result is carried into a broader library. The ledger should distinguish the person or run that produced the local observation from the expert who recognised its natural generality, reconciled it with prior art, designed the library interface, rewrote or formalised the proof, and stewarded review. If that expert turns a problem-specific result into a Mathlib-quality contribution, the project does not demand ownership of the resulting pull request or paper. It asks for a reasonable provenance link back to the route from which the idea arose, while crediting the expert fully for the mathematical judgement and upstream work they actually performed. Provenance is not a device for seizing authorship in arrears.

This policy is deliberately generous because attribution is part of the growth strategy. A stranger needs a reason to spend thought or compute on a project whose open problems remain unsolved. Durable, inspectable credit for partial and negative work is one such reason. It is also technically useful: provenance makes later correction, reproduction, and literature attribution possible.

For a contributor outside a university, research institute, or AI laboratory, the accepted artifact is also something concrete they can point to. The receipt identifies the idea or implementation they supplied, the public paths that carry it, the evidence reviewers checked, the roles they performed, and the boundary that remained. This is not a degree, peer review, employment recommendation, or guarantee that another institution will value the work. It is a durable public account of an actual mathematical or architectural contribution, rather than a claim stranded in a private conversation or an unattributed model transcript.

Record now, trace consequences later

The first ledger should remain descriptive. A contribution receipt records who supplied which object, from which public state, under which role, with which evidence and limitation. It need not guess the contribution's eventual importance. Later accepted work can add reviewed lineage edges such as *uses*, *enables*, *repairs*, *formalises*, *refutes*, or *explains*. A consequence view can then show which later results, tools, or expositions depend on an earlier contribution.

This later view begins with an immediate obligation. Before a result is returned, the clone-local `propagate-research-consequences` skill gives every plausible downstream consumer a disposition. The later graph may reveal further consequences as the corpus grows, but it does not excuse leaving known current consumers stale.

This is more informative than awarding points at intake, but it is not an objective measure of impact. Time does not establish causation, and a graph edge is an authored claim that may need correction. The view should therefore remain inspectable, non-scalar, and corrigible. A small observation may later have many consequences; a large compute run may have none beyond its negative result. Both retain their original receipt.

Any financial reward scheme would require a separate prospective policy for eligibility, conflicts, funding, tax, dispute, and the treatment of old receipts. Existing attribution must not silently become a financial contract. The graph can record that an idea mattered; it cannot, at present, send an invoice.

Pure mining runs can still disclose the model, provider, harness, compute donor, starting commit, and any human direction as distinct roles.

8 Distributed compute without distributed authority

Volunteer compute has an established precedent. BOINC packages scientific jobs for heterogeneous consumer devices, and BOINC Central explicitly aims to make volunteer computing available to scientists without the resources to operate their own project [2]. GIMPS shows a mathematical version: volunteers run a common search program, independent machines confirm a prime, and discovery credit includes the compute donor, software authors, server operator, and wider volunteer effort [3].

Agent-assisted research is less uniform than either example. Most tasks are not interchangeable work units with a predetermined verifier. A runner may change code, propose a conjecture, or misread the problem. The public system therefore distributes *search* while keeping authority local to each evidence type. Mathematicians and formalisation contributors design or review the routes; volunteers may execute them with Claude Code, Codex, Cursor, Antigravity, another open or proprietary harness, or no agent at all. The runner is replaceable. The task packet, evidence boundary, and return record are the shared protocol. Returned code is untrusted until reviewed. Expensive or privileged continuous-integration jobs must not execute fork code with repository secrets. In particular, a GitHub `pull_request_target` workflow must not check out and execute an untrusted fork; GitHub documents that combination as a route to secret leakage and repository compromise [14]. Initial checks should run with read-only permissions, no secrets, bounded resources, and explicit artefact retention. Promotion to trusted infrastructure is a later review decision.

A useful compute return records at least the source commit, model and runner, tool disclosure, prompt or task packet, wall-clock time, human time, token or subscription use, and hardware budget where available, together with commands, changed paths, outputs, validation receipts, failures, and the stopping rule. These inputs should not be forced into one exchange rate: a mathematician's hour, an API bill, and a donated graphics processor are different resources. They can still be disclosed well enough to make a capability comparison interpretable and to reveal when hidden scaffolding paid most of the cost.

The first public mode may remain deliberately simple: contributors clone the repository and run their own agents locally. A later volunteer-compute layer could distribute signed, immutable task packets and receive result bundles without granting write access. It should be built only after task identity, sandboxing, deduplication, resource budgets, result replay, and abuse handling have explicit owners.

9 Precedents and the missing composition

No part of this strategy is without precedent. The project deliberately packages practices developed by the Erdős Problems project, Lean and mathlib, Comparator and Palomar, DeepMind's `formal-conjectures`, Polymath, BOINC, GIMPS, and recent multi-agent formalisation systems. The intended contribution is an accessible composition: a person should be able to enter through one clone without first recreating those infrastructures.

Distributed compute. BOINC and GIMPS show that members of the public will donate hardware to a scientific objective when the client is easy to run, the work is visible, and credit is legible [2, 3]. Their tasks are much more mechanically uniform than open-ended proof research.

Distributed mathematical insight. Polymath projects invite participants at different mathematical levels to share small observations as they occur. Their rules explicitly welcome tentative and negative insights, provided they are made clear enough for others to absorb, and treat the project as collaboration rather than a race [4]. Polymath supplies a social protocol; it does not supply a formal proof, computation, and agent-return substrate for every comment.

Distributed formalisation. Mathlib uses ordinary fork-and-pull-request practice, human review, and source-level attribution. The Carleson formalisation used a public blueprint and many claimable lemma-sized tasks, with formalisation feeding corrections back into the informal plan [5, 6]. Google

DeepMind's `formal-conjectures` repository similarly uses one-problem files, issue assignment, pull requests, metadata, and stable snapshots for a growing Lean benchmark [9]. These projects demonstrate that large checked mathematical objects can be made publicly divisible.

Multi-agent formal research. Agent Hunt studies bounties, locks, guarded ownership, and collaborative agents for autoformalisation [11]. Lean Atlas uses formal dependency information to reduce the declarations a person must inspect for semantic verification [12]. These mechanisms address search, coordination, and review focus. They do not by themselves provide a public commons in which mathematics and improvements to the research architecture enter one provenance-preserving adoption path.

Proof abundance. Tao separates problem solving into generation, verification, exposition, digestion and acceptance, and canonicalisation, and argues that proof abundance will create bottlenecks between these stages [1]. The strategy here treats those bottlenecks as contribution surfaces. A reviewer who prevents an overclaim or an expositor who makes a hard step recoverable is not ancillary to the research pipeline.

The proposed composition joins these precedents. It combines volunteer compute, small shared insights, formal task decomposition, agent-native problem worlds, explicit negative knowledge, untrusted public returns, role-aware credit, and human review. The candidate contribution is this composition and its implementation in one open mathematical corpus. The paper makes no universal priority claim and reports no controlled comparison showing that the composition increases discovery rate.

10 Adding another problem world

The architecture is intended to hold more than the present eight problems, but adding a folder is not enough. A coherent new world needs a canonical question and primary source; a dated account of its current status; an explicit public claim boundary; a readable primer or problem note; formal statements with an informal-faithfulness account where formalisation is appropriate; known literature, computations, counterexamples, and failed routes; exact open contribution points; navigation and validation routes; attribution; and a maintainer or review path.

The public skill for adding a problem separates three states. A proposal may begin as a sourced issue. An incubating formal lane may add checked source under a problem-owned namespace while stating that it is not yet a reviewed public claim. A fully indexed world joins the problem registry, paper corpus, query routes, return schema, cold-start inventory, and relevant external crosswalks. These are distinct transitions because a checked proposition, a published paper, a reviewed claim, and a Comparator selection are different facts.

After any of these transitions, consequence propagation inspects the problem registry, paper corpus, query routes, cold-start inventory, validation roster, return schema, and external crosswalks. A new problem is complete at its stated entry level only when every affected consumer has an explicit disposition; a new directory is not itself a lifecycle receipt.

The last transition is not yet a one-command public operation. The current paper-corpus fan-in has a maintainer-owned stage, several checks enumerate the present roster, and the bounded problem index is already close to its size limit. The immediate infrastructure work is therefore to derive rosters from one public authority, admit an explicit incubating state, split verbose detail out of the bounded index, and make an absent external crosswalk record a normal state rather than an error. Until then, the add-problem skill reports the couplings instead of concealing them behind a scaffold command.

Arbitrary depth is a design target, not a measured scaling result. The useful test is whether a cold agent can retrieve the relevant neighbourhood without reading every paper or rediscovering an existing failure as the number and depth of worlds grow. That test should be repeated whenever the roster or navigation machinery changes.

11 Participation and growth

The project should present several independent reasons to clone the repository.

- A **mathematician** can see eight exact frontiers and contribute a lemma, construction, counterexample, reference, or critique without learning the private orchestration system.

- A **Lean user** can formalise a paper deduction, audit a statement, simplify a proof, or improve the checked interface.
- A **compute hobbyist** can run a local agent against a bounded route in a real open problem and return reproducible evidence even when it is negative. The contributor is not asked to certify a proof claim; mathematical review remains downstream.
- An **agent builder** can compare runners on a stable problem world while disclosing compute, attempts, and starting state.
- An **infrastructure contributor** can improve the conversion from all other inputs to useful mathematical output.
- A **reviewer or expositor** can reconcile meaning, rank results, correct attribution, or turn a checked proof into recoverable understanding.

The lowest-friction invitation should appear on the first screen of the README: clone the project, point a local agent at the compact entry file, and ask it to choose a bounded frontier that matches the available tools. The same screen must say that all eight problems remain open, that a useful return need not solve one, and that architecture contributions receive first-class credit.

Publicity should follow a working contribution loop rather than precede it. A Hacker News launch, research talk, or model-community post can then make a specific claim: a stranger can clone the corpus, reach an exact open question, run or improve the machinery, return a typed result, and see accepted work in public lineage. Stars, clones, and raw agent-hours measure attention or activity. They do not measure mathematical progress.

12 Evaluation

The strategy should be evaluated at each conversion boundary. Useful initial measures include:

- time from a cold clone to a correctly scoped first task;
- fraction of returned bundles that can be replayed at the stated commit;
- fraction requiring statement or evidence-class correction;
- time from return to first substantive review and to adoption decision;
- repeated-dead-end rate before and after a no-go enters the corpus;
- reviewer time per accepted result family;
- contribution diversity across mathematics, computation, software, validation, exposition, and review;
- correction lineage completeness and attribution retention; and
- change in useful output at controlled compute when navigation or another infrastructure component changes.

The numerator must remain claim-bounded. One accepted counterexample may be more useful than hundreds of generated lemmas. Generated certificate shards must not be counted as independent discoveries. A theorem accepted by Lean but rejected on intended meaning is not a successful public transition. Likewise, a strong negative result can improve the corpus even though the endpoint problem remains open.

Longitudinal model comparisons should use immutable problem snapshots and held-out variants where possible. Public hard problems are susceptible to training-data contamination, and the repository itself will become more informative over time. A later model therefore receives both a stronger model and a richer corpus. The design should measure these factors separately rather than presenting every later success as model improvement.

The present work supplies no such evaluation. It establishes an implemented case, an explicit protocol, and falsifiable measures for a later study.

13 Learning from every run

The research memory has three levels. Problem-local memory records the exact objects that change the mathematical search: proved lemmas, counterexamples, finite data, failed mechanisms, and unresolved obligations. Reusable mathematical memory records results that have survived a separate generalisation and integration pass. General process memory records lessons that transfer: a navigation failure, an experiment pattern, a formalisation pitfall, a validation gap, or a review rule. A local theorem does not enter the second level merely because its variables have been renamed.

A continuous mining run should execute a visible loop:

1. re-read the current frontier and the relevant corpus neighbourhood;
2. check whether the proposed route, calculation, or failure is already recorded;
3. compare distinct attacks and run the cheapest exact computation that can distinguish or refute them;
4. carry out the warranted analytic and incremental Lean work;
5. attack the strongest candidate for dropped hypotheses, counterexamples, and stronger prior art;
6. return the smallest evidence-bearing delta with its starting state and surviving boundary; and
7. add problem-local knowledge while proposing reusable process lessons separately for review.

A continuous stewardship run executes a second loop beside it:

1. wake on a stable theorem, counterexample, computation, authority change, paper correction, or review outcome rather than on a timer;
2. rebuild the relevant candidate universe from source authority, including stronger results absent from the present paper or Comparator roster;
3. group declarations into coherent mathematical families and distinguish a new mechanism from a renamed residual, routine corollary, or duplicate;
4. decide proof authority, mathematical significance, exposition order, and future work allocation separately;
5. reconcile the paper, Comparator interface, Palomar disposition, claim boundary, navigation, and contributor lineage wherever the result has a real consequence; and
6. return a source-pinned frontier update: the strongest surviving result, its hard step, the exact boundary, and the next discriminating question.

This second loop explains why paper order is allowed to change as the corpus improves. A paper is a reader-facing projection of current mathematical judgement, not a chronological dump of agent activity. The strongest exact results and mechanisms should lead; routine scaffolding remains available but receives less space. Comparator remains an exact-interface firewall, and Palomar remains an external review route. The stewardship goal may prepare and prioritise those objects, but it cannot award novelty, acceptance, or canonical status to itself.

Subagents can divide literature reading, computation, proof search, formalisation, and adversarial review when their questions and evidence remain independent. The integrating agent must read and verify their returns, and each lane keeps its own starting state and stop condition. Parallel agents can multiply attempts. They do not multiply truth.

The system calls learning a reusable lesson from a local run *up-propagation*. A local lesson is not copied immediately into a universal mathematical-reasoning rule. It is first stated with its evidence, sibling cases, proposed scope, and over-generalisation guard. If the pattern survives review, it may improve a general skill or route. Computational reasoning can likewise become a specialised skill with reusable experiment forms, but every new experiment still records its finite domain and interpretation.

13.1 The local-to-general track

Hard problems are useful partly because they generate mathematical by-products under sustained pressure. A partial theorem that does not solve its parent problem may still deserve a life outside it. The proposed digestion route is:

1. identify the exact local result and retain its problem-specific proof and provenance;
2. separate the hypotheses genuinely used from the coordinates and names of the motivating problem;
3. search the literature and the target library for an existing theorem, collision, preferred abstraction, and likely downstream consumers;
4. state the natural general result, with the local theorem exhibited as a transparent special case rather than hidden by the abstraction;
5. falsify proposed weakenings, check the general proof independently, and test at least one other real use; and
6. ask a qualified mathematician and Lean contributor to decide whether the result is important, well-shaped, documented, maintainable, and ready for an upstream discussion.

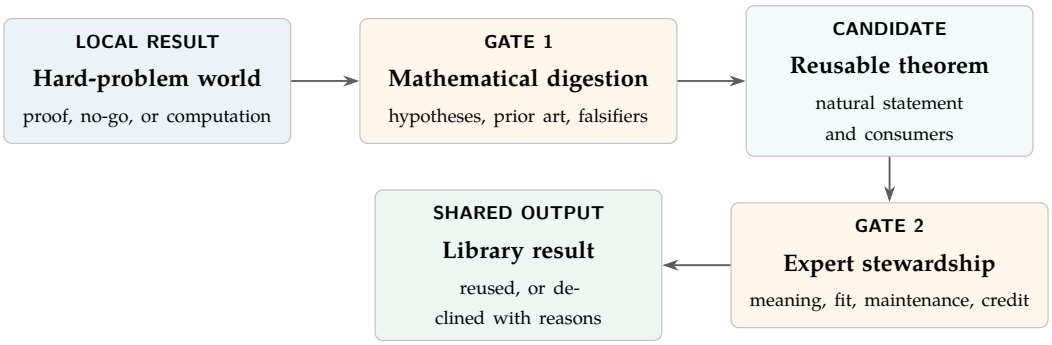

Figure 5: A local result becomes reusable only through a second mathematical and social gate. Formal correctness is necessary where Lean applies, but it does not decide generality, library fit, or stewardship.

Mathlib is the clearest external model for the final stage, and also a warning against automating it badly. Its contribution guide welcomes useful contributions and supplies a public fork-and-pull-request route; it does not reserve contribution to people with institutional credentials. It also sets high standards for generality, integration, maintainability, style, and documentation, and its current AI policy rejects low-quality unsupervised agent submissions while requiring AI use to be disclosed [5]. Accordingly, this project should never direct a compute donor to spray raw agent lemmas into Mathlib. A subject and Lean expert must take custody of a candidate, understand it, decide whether it belongs, bring it to community standards, and own the discussion. The public contribution is then theirs in the roles they performed; the originating problem route remains a provenance fact, not a competing claim of ownership.

The present prototype does not automate this track and reports no Mathlib contribution produced by it. It can preserve local evidence and provenance, and it can help prepare a candidate. Canonical status and upstream acceptance remain external outcomes. The systems paper's mathematical loop describes the authority separation underneath this handoff.

This produces a graph that is richer than a theorem library. Nodes may be endpoints, conjectures, representations, computations, proofs, counterexamples, no-go theorems, reviews, or architecture changes. Edges distinguish proof dependency, implication, equivalence, refutation, obstruction of a method,

experimental support, formalisation, correction, and publication. The edge type matters. A model-generated analogy can nominate a route; it cannot become a proof dependency or a public implication without the corresponding evidence.

As this graph grows, it may support a different kind of agent training or retrieval: successful proofs alongside tempting arguments paired with the exact reason they fail, and theorem statements alongside frontier neighbourhoods showing the remaining cut. Whether this improves mathematical originality or merely makes models more confident within known territory is an open empirical question.

14 A bounded transfer hypothesis beyond mathematics

The same abstract loop appears in experimental science: formulate a hypothesis, design an experiment, observe an outcome, update the theory, and retain both successful and failed interventions. A future agent might define a simulation or experimental script whose variables are linked to a formal model and a semantic account of the hypothesis. Physical observations could then update the research graph and select the next experiment.

Mathematics is only a proof of concept for the *workflow shape*. It is not evidence that the current system can conduct physics, chemistry, or biology. Formal verification has a special strength in mathematics because the target objects and proof rules are digital. In an experimental domain, the instrument, calibration, sample, protocol, measurement uncertainty, physical environment, and causal interpretation become independent authorities. A formally verified control program does not validate the scientific hypothesis or make a laboratory safe.

Any transfer would therefore require domain scientists, controlled physical infrastructure, ethics and safety review, material and environmental limits, and explicit human authority over experiment selection and execution. The initial scope should be low-risk simulation or analysis of existing public data. Autonomous physical experimentation is not a present project goal.

15 Limits and governance

The present strategy has substantial limits.

First, all eight endpoint problems remain open. The repository contains intermediate mathematics and deep records of failed routes, but no evidence that a distributed run will solve any problem. The claim that infrastructure improves the conversion from compute to mathematics has not yet been measured in a controlled experiment.

The prototype had no recorded external user or accepted outside contribution by 31 August 2026. Its clone instructions, agent skills, cross-paper links, pull-request path, and credit machinery may therefore contain friction that an author-operated audit does not reveal. Reproducing a clean clone, reporting a broken instruction, simplifying setup, repairing an unsafe default, or making the contribution route easier to understand are first-class architecture contributions even when they do not alter any mathematics.

Second, the project remains maintainer-centred. The initial problem selection, architecture, claim boundaries, review decisions, and public presentation were all chosen through one operator-controlled process. Open source makes those choices inspectable and contestable; it does not make them independent. A serious growth phase needs external maintainers and reviewers, published conflict and appeal rules, and a way for contributor-controlled reports to remain visible when the project declines to adopt them.

Third, review is scarce. More public agents may worsen the burden by generating plausible but low-value returns. Entry tests, evidence schemas, automated replay, and Palomar-style selection should protect expert attention, but automatic filters cannot award significance or community acceptance.

Fourth, credit is contestable. A role vocabulary preserves more information than a single score, but it cannot mechanically decide authorship, priority, or the relative weight of an idea and its proof. The project must disclose its policy in advance and preserve correction history.

Fifth, an open corpus can be gamed. Contributors may optimise for visible activity, generated theorem counts, or a public model comparison. Strong models may have encountered repository text during training. Security bugs may arrive as apparently useful workflow improvements. No automatic leaderboard should be allowed to become the objective function.

Finally, openness does not remove resource inequality. Compute donors, frontier-model providers, established mathematicians, and maintainers have different access and influence. The strategy can lower the fixed cost of entry and make attribution more durable. It cannot by itself make research participation equitable.

16 Execution order

The strategy should be implemented in the following order.

1. Make the README state the experiment, contributor types, exact open boundary, clone command, agent prompt, return paths, and credit policy.
2. Keep every problem world independently navigable from a cold clone, with one paper-level open section and bounded query routes.
3. Accept both mathematical and architecture returns through typed schemas, human review, immutable generations, and public provenance views.
4. Provide runner-neutral local instructions and safe, unprivileged checks before building a hosted or volunteer-compute scheduler.
5. Recruit external reviewers and maintainers before increasing agent throughput substantially.
6. Run controlled studies of navigation changes, model changes, and compute changes against immutable snapshots.
7. Only after these boundaries work should the project consider a public task market, donated compute service, or transfer to another scientific domain.

The first three steps are partly implemented in the current public clone. Turnkey multi-provider mining, a public volunteer-compute scheduler, independent governance, and a cross-domain laboratory interface are not reported capabilities.

17 Conclusion

The open-source strategy is to make mathematical infrastructure a shared conversion layer. A contributor may add compute, mathematical direction, formalisation, software, review, or exposition without rebuilding the whole system and without pretending to have solved an endpoint problem. Accepted mathematical work enlarges the reusable corpus. Accepted architecture work improves the process that produces and checks later work. Both retain public provenance.

The eight Erdős problems are hard test worlds for this design. They make failure unavoidable enough that the system must learn to preserve it, and deep enough that improvements in models, compute, infrastructure, and human insight can be observed over time. Their role is not to guarantee a dramatic solution. It is to ensure that the research commons is exercised on real mathematics with exact open boundaries.

The strongest claim at present is therefore architectural. A fresh clone can carry a stranger from a hard question to a bounded frontier and give that stranger a path for returning evidence and receiving credit. The next test is social and empirical: whether outside contributors can use that path, whether reviewers can absorb the returns, and whether the accumulated graph makes the next attempt genuinely better.

A Public entry routes

The public repository is `wcook04/plectis-lean-erdos249-257`. The shortest current routes are:

Question	Public route
What is the experiment?	`README.md`
Where should an agent begin?	`AGENTS.override.md`
How can an agent explain the system?	`explain-public-system skill`
How can an agent run the coupled research lifecycle?	`coupled-goal skill`
How can an agent mine a frontier?	`mine-open-problem skill`
How is a bounded Lean change validated?	`lean-concurrent-validation skill`
How are downstream consequences reconciled?	`propagate-research-consequences skill`
How are clone skills installed elsewhere?	`install-clone-skills skill`
What is proved and what remains open?	`docs/RESULTS.md`
Which papers and problems exist?	`docs/papers/README.md`
How can I contribute mathematics?	`CONTRIBUTING.md` and the research-progress issue form
How can I improve the architecture?	`architecture contribution guide` and the architecture-proposal issue form
How can an agent prepare a pull request?	`submit-pull-request skill`
How can I propose or add another problem?	`add-open-problem skill`
How is a return validated?	`erdos-research-return skill` and the `research-commons protocol`
How is credit recorded?	`credit policy`
What do Comparator and Palomar establish?	`Comparator guide` and `Palomar qualification`

Declaration of generative AI use

Every word of this manuscript was generated by agents based on large language models operating within Will Cook's private research system for artificial intelligence. The formal proofs and repository software were likewise drafted and revised by the agents through that system under Cook's direction. Cook set the objectives and acceptance criteria, selected and reviewed the public claims, and approved the published version. Cook assumes responsibility for the accuracy, interpretation, and presentation of the work. Generative systems are production tools, not authors, and supply no independent authority.

References

[1] T. Tao, *Mathematics in the age of AI*, 2026, arXiv:2608.16753.

[2] D. P. Anderson, *BOINC: A Platform for Volunteer Computing*, Journal of Grid Computing 18 (2020), 99–122, DOI.

[3] Great Internet Mersenne Prime Search, *GIMPS*, project documentation and discovery-credit record, mersenne.org, accessed August 2026.

[4] Polymath Project, *General polymath rules*, project rules, accessed August 2026.

[5] Lean community, *Contributing to mathlib*, contributor guide, accessed August 2026.

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

[7] T. F. Bloom, *Erdős Problems*, problem database, sources, and forum, erdosproblems.com, accessed August 2026.

[8] L. de Moura and S. Ullrich, *The Lean 4 Theorem Prover and Programming Language*, Automated Deduction—CADE 28 (2021), 625–635, DOI.

[9] Google DeepMind, *formal-conjectures*, software repository, accessed August 2026.

[10] Palomar Registry, *About Palomar* and *Contribution policy*, registry documentation and submission standard, accessed August 2026.

[11] C. E. Brown, C. Kaliszyk, and J. Urban, *Agent Hunt: Bounty Based Collaborative Autoformalization With LLM Agents*, 2026, arXiv:2603.06737.

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

[13] NISO, *CRediT: Contributor Roles Taxonomy*, role definitions, accessed August 2026.

[14] GitHub, *Preventing pwn requests*, GitHub Actions security guidance, documentation, accessed August 2026.