The GitHub Paradigm

Human to Bot, Bot to Bot, and the End of Attribution

Douglas C. Youvan

2026

Copyright and Publication Note

This book is an interpretive study of human-machine research workflows, repository history, software provenance, authorship, and attribution. It discusses public repository artifacts and publicly documented platform features. Descriptions of AI contribution are descriptions of workflow roles, not claims that current machine systems possess legal authorship, personhood, or moral responsibility.

Repository and platform features described as current were checked during preparation in August 2026. Online systems change; readers should consult current documentation for operational details.

Contents

Preface - Why This Book Exists

This book grew out of a practical question rather than a theoretical one: after a sequence of human-AI research projects had been turned into public GitHub repositories, what did the repository history actually say about who created them? The answer was unsettling. The artifacts were increasingly explicit about mathematical assumptions, verification, package structure, exact outputs, and release lineage, yet the conventional authorship record could remain little more than the name of the account that uploaded the files. The technical record was becoming more rigorous while the intellectual record was becoming less descriptive.

That mismatch is not unique to one research program. GitHub is rapidly becoming a workspace for coding agents, automated review, security scanning, continuous integration, and third-party machine agents. Human-to-bot delegation is already ordinary. Bot-to-bot influence is becoming ordinary through pull requests, reviews, patches, tests, and persistent repositories. The platform that perfected distributed artifact lineage is therefore becoming the obvious place to ask a new question: how should we preserve the lineage of ideas when the actors are heterogeneous and some of them are transient machine sessions?

The argument is not that machines should simply be added to human bylines. Nor is it that humans cease to matter because a model generated many of the words or lines of code. The argument is that 'authorship' now compresses too many distinct relations. Direction, generation, criticism, verification, selection, release responsibility, legal rights, and historical origin can belong to different actors. A flat attribution cannot represent the resulting graph.

The case studies in the middle of the book are drawn from public repositories because they expose the workflow at unusually high resolution. They show projects that evolved through multiple releases, were attacked by independent reviews, narrowed their novelty claims, replaced approximate computations with exact ones, stopped branches after negative theorems, and spawned conceptual descendants. The purpose is not to use those projects as proof that any particular model is autonomous. It is to show that the artifact-centered workflow already creates forms of intellectual inheritance for which our attribution vocabulary is inadequate.

The proposed solution is modest: preserve semantic contribution events alongside Git history. Record consequential human steering, machine generation, adversarial review, verification, rejection, generalization, and release. Do not publish private chain-of-thought. Do not archive every token. Do not confuse causal provenance with legal authorship. Instead, build a small event graph that lets future readers reconstruct how the project changed and why.

Method Note - How the Corpus Was Read

The repository corpus in this book is used as a documentary record, not as a complete transcript of cognition. A repository can establish that a particular file existed in a particular release, that a README made a particular claim, that tests or certificates were shipped, that one version declared another as its parent, or that a release changed its assumptions and claim boundaries. It can often show the architecture of a research program with remarkable precision. It cannot, by itself, establish which sentence was first imagined by a person, which proof move was proposed in an AI conversation, which criticism came from a separate model session, or which discarded alternatives never reached the public artifact. Those are different evidentiary questions.

Accordingly, the case studies are read at three levels. The first level is artifact evidence: repository names, version labels, README statements, source files, manifests, tests, result files, and explicit parent-child declarations. These are public objects that can be inspected independently. The second level is workflow evidence: the observable pattern by which projects acquire independent verifiers, narrowed novelty boundaries, corrected arithmetic, stronger reproduction contracts, or successor repositories. Such patterns support claims about how the research process was organized even when they do not identify the exact origin of every idea. The third level is participant recollection: descriptions of human prompting, AI generation, firewalled review, rejection, and redirection. Those descriptions are useful, but they are not silently elevated into properties proven by Git itself.

This distinction matters because the thesis of the book would become circular if machine agency were inferred merely from the existence of machine-produced-looking artifacts. The argument does not require a claim that a model is a legal author, a conscious scientist, or an autonomous moral agent. It requires only the more limited observation that modern research workflows can distribute causally important operations across humans and machine systems: proposing text or code, finding defects, generating tests, comparing prior art, revising proofs, and producing descendants. Whether those operations deserve the word authorship is precisely the question under examination.

The repository names are therefore treated as nodes in a genealogy rather than as proxies for individual minds. A versioned descendant can preserve, reject, or transform properties of a parent artifact. A new repository can inherit a question from an earlier branch while changing the mathematical object completely. A verifier can become part of the durable research object even though the reviewing model session that motivated it disappears. This is the sense in which bot-to-bot transmission can occur without direct communication between two persistent agents: one machine leaves an artifact; another machine later reads, criticizes, transforms, or generalizes that artifact. The repository mediates the inheritance.

The book also distinguishes current platform capabilities from speculative extensions. When it describes coding agents, pull-request review, co-author metadata, or provenance standards, it relies on public documentation current during preparation in August 2026. When it imagines autonomous literature review, repository-to-repository research ecosystems, or standardized intellectual-provenance ledgers, those passages are proposals or forecasts and are labeled accordingly. The purpose is not to blur present capability with possible future capability, but to show that the conceptual problem already exists before the most ambitious future arrives.

Finally, the corpus is not offered as a statistically representative sample of all GitHub use. It is a dense longitudinal case study. That limitation is also its advantage. A conventional survey might show how often developers use AI; it would reveal less about what happens when the same human-machine research process repeatedly produces theorem packages, audits, negative results, repairs, branch terminations, visual demonstrations, calculators, controls, and generalizations. The corpus lets us follow those transformations closely enough to ask what ordinary commit history preserves and what it loses. That is the methodological role it plays throughout the book.

PART I - THE RECORD WE THOUGHT WE HAD

Chapter 1 - Add Files via Upload

It is the central problem of authorship.

The deceptively simple commit

A repository can contain a theorem, a program, an experimental protocol, a critique, a set of regression tests, and enough provenance to reproduce every byte of the release. Yet the visible history can reduce the decisive moment to a sentence as thin as "Add files via upload." That phrase is not false. Files were uploaded. It is simply answering a much smaller question than the one a reader usually thinks a historical record is answering. The commit tells us who performed a repository operation. It does not necessarily tell us who conceived the problem, who proposed the architecture, who generated the first proof, who found the defect, who rewrote the proof after criticism, or which machine instance supplied the conceptual bridge from one project to the next. In an AI-assisted research workflow, the difference between those questions is no longer a technicality. It is the central problem of authorship.

Git was built to make changes inspectable and lineage recoverable. GitHub made that lineage social: branches could be proposed, discussed, reviewed, merged, forked, compared, and attributed to accounts. For most of the history of collaborative programming, the account and the intellectual actor were close enough that ordinary practice could tolerate the distinction. A commit made under Alice's identity was presumed to represent work Alice either wrote or deliberately adopted. A co-author trailer could add Bob. A pull request could reveal a conversation among named people. That model was never philosophically perfect, but it was operationally adequate. Generative systems break the adequacy without breaking the repository. The repository continues to function while the meaning of its historical metadata changes underneath it.

This book begins with that mismatch. It is not an accusation that GitHub has failed. GitHub is doing what it was designed to do, and it is rapidly adding explicit coding agents and AI review mechanisms. The claim is instead that a platform designed around the ancestry of artifacts is becoming infrastructure for a world in which the ancestry of ideas may follow a different graph. Once humans can hand questions to models, models can hand artifacts to other models, and those models can review, repair, extend, or fork the work, the commit graph and the intellectual graph can diverge radically. The GitHub paradigm is the moment when that divergence becomes normal rather than exceptional.

A natural experiment in authorship

The repositories examined in this book provide a useful natural experiment because they were not created to demonstrate a theory of authorship. They were created to do mathematics, build

research software, test ideas, repair defects, and deposit stable releases. The authorship problem appeared afterward, as a property of the workflow itself. A human might specify a direction such as moving away from a familiar physical interpretation, ask for a computational construction, reject an uninteresting branch, demand an adversarial review, decide that a version should not be deposited, and then choose the next conceptual move. Meanwhile an AI system might write thousands of lines of code, derive lemmas, design tests, draft documentation, run verifiers, identify flaws, and propose new theorems. Another isolated AI instance might attack the package and force a revision. The final GitHub repository records the product far more completely than it records that division of intellectual labor.

The striking feature is not that a machine can produce code. Automated code generation is already ordinary. The feature is that the machine-generated artifact can become a durable intellectual object that another machine later treats as prior work. That second machine does not need the first machine's identity. It needs the repository. The artifact becomes the interface. Once this happens repeatedly, bot-to-bot transfer can occur even when no persistent bot identity survives. The machine actors can be transient while the artifact lineage is permanent. GitHub, without needing to represent minds at all, can become a medium through which machine-produced intellectual structures reproduce.

That possibility suggests the title's provocative phrase, the end of attribution. The phrase should not be read as a prediction that names will disappear from books, commits, or papers. Names may proliferate. The stronger claim is that attribution may cease to be an adequate account of origin. We may know the publisher, depositor, maintainer, legal author, repository owner, model provider, reviewer, and final editor and still fail to know where the key idea came from. The future could produce better artifact provenance than any previous era while simultaneously producing weaker intellectual provenance.

Further implications

The phrase "Add files via upload" also illustrates a subtle archival inversion. In earlier scholarship, the intellectual act was often richly narrated while the physical production history was poorly preserved. We might possess correspondence explaining why a theorem mattered but lack exact computational inputs. In repository science the opposite can happen: every byte is preserved, but the decisive conversation vanishes. This inversion means that future historians cannot assume digital abundance equals historical completeness. A repository can be forensically exact and intellectually mute about its own genesis.

The remedy is not to turn every commit message into autobiography. Commit history should remain operational. The missing information belongs in a parallel layer whose granularity is driven by consequence. If a review changed a theorem from universal to conditional, that deserves a record. If a human rejected an attractive but unverified interpretation, that deserves a record. If a

later model turned a one-off obstruction into an infinite family, that deserves a record. The result would be a history that respects both software engineering and intellectual history instead of asking one metadata field to perform both jobs.

Chapter 2 - What Git Was Built to Remember

> *A commit message can explain intent, yet Git itself does not model intent as a first-class object.*

A history of states, not a history of minds

Git is extraordinarily good at a problem that had become painful by the early twenty-first century: how to preserve and compare the evolving state of a distributed software project without making a central server the sole source of truth. A commit identifies a tree of content and connects it to one or more parents. Hashes make tampering visible. Branches permit divergent development. Merges join histories. Tags identify important states. Every clone can contain the complete history. This is a profound architecture for distributed memory, but its unit of memory is the artifact state. A commit message can explain intent, yet Git itself does not model intent as a first-class object.

The distinction matters because humans naturally read technical history as biography. We see a name on a commit and infer an actor. We see a sequence of commits and infer a sequence of thoughts. We see a branch and imagine a decision to explore one possibility rather than another. Often those inferences are good enough. But they are extra information supplied by our social assumptions, not by the content-addressed graph itself. The graph says that state B descended from state A. It does not prove that the named committer personally typed the changed lines, understood every implication, or originated the conceptual plan.

Even before generative AI, this gap existed. A developer might commit generated parser code, vendor a library, apply a patch written by someone else, copy a snippet from documentation, or squash twenty authors' work into one commit. Teams developed conventions to compensate: sign-offs, co-author trailers, pull-request discussions, issue references, code review, contributor files, release notes, and licenses. These practices add social context around the artifact graph. They work because humans recognize the need to restore meaning that the core version-control object does not encode.

Why the old approximation worked

The old approximation worked because the cost of producing a coherent technical artifact was usually correlated with sustained human involvement. A source file might include generated portions, but a human team typically selected the architecture, integrated the code, understood the requirements, and accepted responsibility for the result. When the repository said a person committed a change, the mismatch between operational attribution and intellectual contribution

was bounded enough to be socially manageable. Academic software complicated the picture, but not usually beyond recognition.

Generative systems alter the scale of the mismatch. A single human instruction can lead to hundreds of files, tests, documentation pages, data structures, and explanatory prose. A second model can inspect those outputs and propose corrections. A third can generalize the corrected result. At that point, the repository owner may be the decisive intellectual director without being the line-by-line author. Conversely, a model may generate most of the surface artifact without having a persistent identity or legal standing. The traditional categories - author, programmer, reviewer, maintainer - no longer align neatly with the causal production process.

Git therefore gives us a useful baseline concept: artifact lineage. It answers which recorded state descended from which prior recorded state. The central argument of this book is that AI-era research needs a second graph layered on top of that one: contribution lineage. Contribution lineage asks which human decision, machine session, prior artifact, review, external source, or automated verifier materially caused each important transformation. The two graphs need not be identical. Indeed, the more capable agentic systems become, the more dangerous it will be to pretend that they are.

Further implications

Git's success can itself mislead us because content addressing feels like complete memory. A commit hash is an extraordinarily strong identifier for a particular state, and parent links make ancestry precise. Yet precision of identification is not precision of explanation. The fact that we can prove file B descended from file A does not prove that the idea in B descended only from A. A model may have consulted a paper, another repository, or an earlier conversation while producing the patch. The content graph can therefore understate semantic inputs even when it perfectly records textual ancestry.

This distinction suggests a useful design test. Ask whether a future reader could reproduce the artifact and separately whether the reader could reconstruct the main reasons the artifact has its current form. Git is optimized for the first question. Research provenance must answer the second. The two answers reinforce each other but should not be conflated. A project that can reproduce results but cannot explain why its claim boundary changed is technically reproducible yet historically incomplete.

Chapter 3 - GitHub Makes the Graph Social

> *The result became not merely version control but a public theater of software production in which identity and artifact change appeared together.*

From commits to conversations

GitHub did more than host Git repositories. It placed the commit graph inside a social environment. Pull requests transformed a branch into a proposal. Issues made problems addressable objects. Reviews made criticism part of the visible development record. Forks made derivation socially legible. Profiles, contribution graphs, organizations, releases, Actions, discussions, and project boards added layers of coordination. The result became not merely version control but a public theater of software production in which identity and artifact change appeared together.

That social layer is why the AI transition is especially consequential on GitHub. An agent does not need a new storage medium. It can enter the same architecture humans already use. GitHub's current coding-agent features make this explicit: a task can be assigned to an agent; the agent can work on a branch and create a pull request; a reviewer can comment; and the agent can iterate. Copilot can also perform code review, including automatic review, and GitHub supports third-party coding agents working in the same pull-request environment. The platform is therefore evolving from a social coding network for humans into a mixed network of humans and machine agents.

This is not merely a user-interface change. The pull request becomes a boundary object that can be consumed by either kind of actor. A human can read it. A model can read it. Tests can judge it. Security scanners can inspect it. Another agent can repair it. The same artifact is simultaneously code, message, evidence, task specification, and input to another computational actor. That makes the repository a natural habitat for machine-to-machine intellectual exchange even when the system was originally optimized for human collaboration.

Attribution survives, but changes meaning

GitHub already has mechanisms for explicit attribution. A commit may list co-authors through Co-authored-by trailers. Bot accounts can be recognizable actors. Agent-created pull requests can be labeled and reviewed. These are important features, and a serious argument about attribution must acknowledge them. The issue is not that GitHub records nothing. The issue is that the recorded identities usually correspond to repository events, while the causal path of an idea can cross systems that are not represented as durable identities in the repository.

Suppose a human asks one model to formulate a theorem. The model produces a package. A second model independently reviews it and finds a counterexample. The human instructs a third model to repair the theorem without returning to the discredited assumption. The third model produces a narrower result. A fourth model notices that the repair can be generalized from prime fields to prime-power fields. The human deposits the resulting release. Even if every final file is perfectly versioned, which actor should appear in the author field? There may be no single answer, because 'author' is compressing several different relations: originator, generator, critic, selector, guarantor, depositor, and legal claimant.

The GitHub paradigm does not abolish these relations. It exposes their plurality. A future repository may need to say not merely 'Alice authored this commit' but 'Alice supplied the research objective; model session M1 drafted theorem T; model session M2 falsified lemma L; Alice rejected architecture A; M3 produced repair R; verifier V certified tests; Alice accepted and deposited release 0.3.1.' That record is much richer than conventional attribution, but it is also much closer to what actually happened.

Further implications

Agentic GitHub features make the distinction between actor and account increasingly explicit. An issue can be the task object, a machine agent can produce a pull request, a different machine system can review that pull request, automated checks can reject it, and a human maintainer can merge the result. Each stage is visible in the platform, but the conceptual contribution of each stage may still be implicit. The infrastructure already contains most of the nodes we need; what is missing is a vocabulary for the semantic role of the edge between them.

A future interface could therefore display more than 'authored by' and 'reviewed by.' It could display 'generated from issue,' 'independently reviewed,' 'claim narrowed after review,' 'tests added in response,' or 'conceptually descended from release.' Those labels would turn GitHub's social graph into a research graph. The technical challenge is modest compared with the cultural challenge: communities would have to decide which events are worth declaring and how much confidence to attach to self-reported provenance.

Chapter 4 - The Old Social Contract of Authorship

Disputes about order, ghost authorship, honorary authorship, and contributor thresholds were disagreements inside a human social system.

Authorship as a bundle of roles

Authorship has always been a bundle rather than a single act. In scholarship it can mean conceiving the problem, designing the experiment, collecting data, analyzing results, writing prose, revising the manuscript, taking responsibility, and receiving credit. In software it can mean architecture, implementation, testing, review, documentation, maintenance, or release management. The bundle was manageable because the roles were carried by people and institutions that could negotiate credit. Disputes about order, ghost authorship, honorary authorship, and contributor thresholds were disagreements inside a human social system.

AI inserts actors that can perform substantial components of the bundle without fitting its normative assumptions. A model can draft a proof but cannot ordinarily accept legal responsibility. It can discover a bug but may not persist as an identifiable entity. It can generate a novel implementation, yet the model provider did not necessarily direct that implementation and the user may not have specified its details. A human can exercise decisive judgment without writing the artifact. Existing vocabulary pushes us toward false choices: either the human is the author of everything or the machine is an author in the human sense. The more accurate description may be a structured contribution graph in which different causal roles remain distinct.

This is why the title speaks of the end of attribution rather than the end of credit. Credit may remain useful and ethically necessary. Attribution, however, is a mapping from an artifact to named sources. When production becomes distributed across ephemeral model sessions, persistent repositories, automated checks, and human steering, a flat name list loses explanatory power. The problem resembles trying to describe a computer network by naming only the owner of the router.

Responsibility cannot be outsourced to opacity

One tempting response is to say that because a machine cannot be morally responsible, the human who publishes the artifact should simply be treated as its author. That may be a sensible rule for accountability, but it does not solve the historical problem. Responsibility and origin are different dimensions. A laboratory director can be responsible for a result without personally pipetting every sample. A journal editor can be responsible for a publication decision without originating the claims. A release maintainer can own the decision to publish code written by many

contributors. AI makes these distinctions unavoidable because the gap between direction and surface production can become enormous.

Another tempting response is to assign authorship to the model name: GPT, Claude, Copilot, Gemini, or another system. That also collapses too much. A model family is not a session, and a session is not the training process that produced the model. The same model can generate contradictory outputs under different prompts and contexts. The relevant causal object may be a particular interaction history, tool environment, artifact set, and model version. Simply writing 'with AI assistance' can be true while hiding nearly everything a future historian would want to know.

The practical conclusion is modest but powerful: record the roles separately. Who set the goal? Who or what generated the candidate? Who or what tested it? Who criticized it? Who changed the claim boundary? Who accepted the risk of publication? Who deposited the artifact? Who can reproduce it? Once these questions are separated, the apparent paradox of machine authorship becomes less mysterious. The challenge shifts from deciding whether a bot is an author to designing a provenance system that can describe mixed human-machine intellectual work without pretending that all contributors are the same kind of entity.

Further implications

Human scholarship already contains precedents for role separation. Large physics collaborations can publish with thousands of names; biomedical journals increasingly request contributor-role statements; software projects distinguish maintainers from contributors; peer reviewers influence manuscripts without byline credit. These precedents show that authorship has long been straining under complex production systems. AI does not create the problem from nothing. It makes the insufficiency impossible to ignore because one nonhuman system can move among several roles within minutes.

The result may be a shift from authorship as identity to authorship as governance. A book can still bear a human author's name because a human shaped the project and assumes responsibility for the work as a whole, while a provenance note records that generative systems drafted, analyzed, or reviewed substantial portions. This is neither concealment nor surrender of authorship. It is a more explicit division between public responsibility and causal production.

PART II - HUMAN TO BOT

Chapter 5 - The Instruction Becomes an Instrument

They specify a space of acceptable intellectual motion.

From prompt to research direction

The phrase 'prompt engineering' is too small for some forms of human-to-bot research. A prompt can be a request for a slogan or a paragraph, but it can also be a constraint on a research program: move away from a particular theoretical framework; preserve the verified results of the previous release; do not claim novelty for classical ingredients; seek a generalization rather than another finite example; keep the runtime dependency-free; or firewall the reviewer from development history. Such instructions do not specify the final artifact line by line. They specify a space of acceptable intellectual motion.

In that sense, the human instruction becomes an instrument. It changes what the machine is allowed to notice and optimize. A demand for an independent verifier creates a second epistemic path. A ban on relying on familiar named structures can force a derivation from primitive data. A request to move away from Copenhagen quantum mechanics can reject an entire family of interpretive shortcuts while leaving the machine free to search for relational mathematics. A decision not to deposit a version preserves the project in a provisional state and changes what later work may legitimately cite as its parent.

The most consequential human act may therefore be neither writing nor prompting in the narrow sense. It may be governance. Governance determines which objectives matter, which claims are permitted, which failures invalidate a release, which directions are abandoned, and which discoveries are promoted into durable artifacts. In a high-output AI workflow, governance can become the scarce intellectual resource. The machine can generate alternatives faster than a person can responsibly evaluate them. Selection becomes constitutive rather than clerical.

The asymmetry of abundance

Generative systems create an asymmetry between proposal and acceptance. Proposals can be cheap and numerous. Acceptance remains expensive because it consumes attention, judgment, domain knowledge, and reputational risk. This asymmetry changes the meaning of authorship. If a model can produce ten plausible architectures in the time a human can deeply examine one, then the human contribution shifts toward constructing filters. The person who says 'this direction is trivial,' 'this theorem overclaims,' 'this is too close to prior art,' or 'continue from this negative result' can exert more influence on the final research program than the actor that wrote the largest number of tokens.

This is familiar in other forms of creative work. A film director may not operate every camera. An architect may not place every brick. A principal investigator may not execute every assay. But AI increases the ratio between direction and produced surface form so dramatically that traditional measures of contribution become misleading. Counting lines, words, or commits can assign most of the visible work to the machine while missing the human's control over the search trajectory.

The GitHub paradigm therefore needs a vocabulary for steering acts. A steering act is an intervention that materially changes the future artifact lineage without directly supplying the artifact content. Rejecting a version, narrowing a novelty claim, demanding a second verifier, changing the target audience, or choosing one branch over another are steering acts. They are not equivalent to authorship, but neither are they mere administration. In mixed human-machine research they can be among the highest-leverage intellectual contributions.

Further implications

The most informative prompts in research often look less like requests for content and more like experimental controls. 'Do not use analogy to familiar structures,' 'treat the package as a fresh submission,' 'move toward novelty and away from Copenhagen,' or 'do not deposit this version' constrain the process rather than dictate the answer. Such instructions can be compared across runs. They can even be versioned. A research group could treat prompt-level constitutions as methodological artifacts much as laboratories treat protocols.

Once prompts are understood this way, provenance should distinguish transient instructions from durable policy. A one-time request to rename a variable is not equivalent to a standing rule that all novelty claims must separate classical ingredients from candidate-new synthesis. Durable policy deserves explicit storage in the repository because it shapes many descendants. It is one of the mechanisms by which a human research style becomes inheritable by future model sessions.

Chapter 6 - Typing Is Not the Measure of Thought

Keystrokes are an implementation detail of expression, not a reliable unit of thought.

The keyboard illusion

Modern authorship inherited a quiet keyboard illusion: the person who physically produces the words or code is presumed to be the principal intellectual source. That assumption was already weak in dictation, collaborative editing, laboratory science, and industrial software. Generative AI makes it visibly untenable. A person can type a ten-word instruction that triggers a thousand-line implementation, while another person can type a thousand words of routine documentation that adds little conceptual content. Keystrokes are an implementation detail of expression, not a reliable unit of thought.

This does not mean that every prompt is profound. Most are not. The U.S. Copyright Office has emphasized that merely supplying prompts does not automatically make a person the author of the expressive elements produced by a generative system. That legal conclusion and the argument here can coexist. Copyright asks whether human expressive control is sufficient for protection. Intellectual provenance asks what causal contributions produced the artifact. A prompt can be legally insufficient for authorship while still being historically important. Conversely, a legally recognized human author can publish an artifact whose conceptual ancestry includes extensive machine-generated structure.

The central mistake is to force legal authorship, scholarly credit, causal origin, and technical provenance into one label. These systems answer different questions. Legal authorship allocates rights and responsibilities. Scholarly credit allocates recognition. Causal provenance reconstructs how an idea developed. Artifact provenance reconstructs how files were produced. A robust GitHub-era record should permit these answers to diverge.

Selection as expression

Where human contribution often becomes unmistakable is selection among machine-generated alternatives. Consider a model that proposes five theorem statements, three computational architectures, or several explanatory metaphors. Choosing one is not automatically authorship of the rejected material, but the sequence of choices can create a recognizable intellectual program. Selection becomes more expressive when it is guided by persistent criteria: exact verification over plausible simulation, narrow novelty claims over promotional language, non-Copenhagen constructions over interpretive familiarity, or repository releases only after adversarial review.

In such workflows the human may gradually train the process without training the model. The model weights remain unchanged, but the project acquires a local constitution. Release conventions, claim boundaries, review rules, input assumptions, and preferred proof styles accumulate. The constitution can be encoded in repository files, prompts, checklists, or simply repeated human interventions. A later model instance can inherit that constitution from artifacts even if it has no memory of earlier conversations. This is one mechanism by which human-to-bot work becomes bot-to-bot work.

The repository thus functions as externalized project memory. It can carry forward decisions that no model session remembers and that the human no longer needs to restate. Once the artifact contains its own assumptions ledger, tests, limitations, related-work boundary, and release policy, the next machine can enter the project at a higher level. The human's earlier steering becomes encoded in the environment. That is a form of durable influence that conventional commit attribution barely captures.

Further implications

This argument also explains why simple quantitative disclosure can be misleading. Saying that 'AI wrote eighty percent of the words' sounds precise but may tell us little about intellectual contribution. A model can expand an outline into fluent prose while the conceptual structure remains human. In another case, a model can contribute one short lemma that unlocks the entire project. Token percentages capture surface production, not causal importance. The same is true of code-line percentages.

A better disclosure names functions. Did the system propose the core idea? Draft the implementation? Search prior art? Produce counterexamples? Review another model's work? Explain results for publication? Each role can be recorded without pretending that contribution can be reduced to a single fraction. Quantification may still be useful for some operational purposes, but the provenance graph should remain primary.

Chapter 7 - The Human as Governor, Editor, and Veto

> *In the repository record, that negative decision may leave almost no trace unless it is documented explicitly.*

The power to stop

One of the most underestimated forms of intellectual control is the power to stop. A machine can continue indefinitely: extend the parameter range, add features, generalize a theorem, generate another visualization, or draft another interpretation. Research quality often depends on refusing continuation. A branch may have reached a theorem that shows why the architecture cannot deliver the foundational objective. The correct move is then not version 0.3.3 with more examples, but termination and a new architecture. In the repository record, that negative decision may leave almost no trace unless it is documented explicitly.

The Relational Constraint Dynamics lineage offers a clean example of this logic. The project classified fixed graphs under a self-generated exact Ising-correlation reconstruction operator and then identified a structural obstruction: independent-component factorization prevents the operator from uniquely deriving one connected world. The README does not merely report a result; it says why the branch should stop and what kind of architecture should come next. That decision is intellectually rich because the negative theorem becomes a routing instruction for future research. A human who recognizes that significance and refuses cosmetic continuation is shaping the research program at a level that line counts cannot measure.

Veto power also applies to publication. A release can be mathematically interesting and still be withheld because a reviewer found an overclaim, a parsing defect, a prior-art issue, or a provenance failure. The difference between 'generated' and 'deposited' is therefore meaningful. GitHub records the deposited artifact very well. A full intellectual history should also preserve rejected versions and the reasons for rejection, because those rejections often explain why the final theorem has the shape it does.

Editorial compression

The governor role is paired with editorial compression. A model may explain a result in ten directions. The human may decide that only two belong in the claim. A model may suggest a sweeping philosophical interpretation. The human may confine the release to a finite theorem and leave the philosophy for a book. A reviewer may note that a classical theorem already explains half of the observed atlas. The next release can then separate established mathematics from the

genuinely candidate-new contribution. This is not merely stylistic editing; it changes the epistemic content of the artifact.

In traditional publishing, editorial labor is often invisible even when it strongly shapes the final text. AI-era research makes invisibility more dangerous because machine output can appear polished before it is well calibrated. The polished surface encourages false confidence. A rigorous human-machine workflow must therefore make subtraction prestigious. Removing a claim, narrowing a scope, documenting a limitation, or declaring a branch exhausted are positive research acts.

The GitHub paradigm can support this if repositories treat claim boundaries as first-class files. A CLAIM_BOUNDARY.md, assumptions ledger, release verification report, or independent-review summary can preserve the decisions that made the artifact defensible. The author's name then becomes less important than the visible structure of judgment. Instead of asking only who wrote the result, a future reader can ask how the result survived.

Further implications

Governance becomes even more important when model output is persuasive. A polished theorem statement, clean code, and passing tests can create the impression that the project is finished. But the remaining questions may be the hardest ones: Is the theorem actually new? Are the assumptions the ones we intended? Did the tests merely encode the same mistake as the implementation? Does the physical interpretation overreach the mathematics? These are often boundary questions rather than construction questions.

The power to withhold a release is therefore a form of epistemic capital. It preserves the distinction between exploratory output and public claim. In a high-throughput AI workflow, that distinction can disappear if every generated artifact is immediately pushed to a public repository. A deliberate deposit gate allows criticism, prior-art checking, and conceptual reconsideration to occur before the artifact acquires the social status of a release.

Chapter 8 - When the Bot Becomes a Laboratory Partner

> *At that point the interaction resembles laboratory partnership, although the machine remains different from a human colleague in agency, persistence, responsibility, and social status.*

Beyond autocomplete

The phrase 'AI assistant' suggests a subordinate tool that accelerates tasks already understood by the user. That description fits autocomplete, boilerplate generation, and routine refactoring. It fits less well when the system derives a counterexample the human did not anticipate, discovers a relation between two mathematical structures, constructs an independent verifier, or proposes the next research object after a negative result. At that point the interaction resembles laboratory partnership, although the machine remains different from a human colleague in agency, persistence, responsibility, and social status.

The practical test is surprise with accountability. If the machine can produce results that were not already mentally specified by the user, then it contributes exploratory capacity. But if those results are accepted merely because they are surprising, the workflow becomes credulous. A laboratory partner must be interrogated. The package needs tests. The theorem needs a proof boundary. The data need provenance. The reviewer needs independence. A claim that survives these constraints has more epistemic value than a fluent answer in a chat window.

GitHub is useful precisely because it converts ephemeral machine output into inspectable artifacts. Code can be run. Tests can fail. Hashes can be compared. A README can overclaim and later be narrowed. A result JSON can be regenerated. The bot stops being a conversational oracle and becomes one participant in an artifact-centered experiment.

The repository as common ground

Human and machine actors do not need the same cognitive architecture if they share a sufficiently explicit artifact. A person can understand a theorem conceptually while a verifier checks finite cases mechanically. A model can read a manifest and reason about missing files. Another model can inspect a proof and compare it with the code. A continuous-integration system can execute tests without understanding the mathematics. The repository coordinates these heterogeneous forms of competence.

This is a deeper reason that GitHub may matter to AI intellectual history. It is not only a place to store machine-written code. It is a common substrate on which different kinds of agents can act without sharing an internal language. The repository is external memory, protocol, evidence, and

negotiation surface at once. The more explicit the files become, the less important it is that the participants remember one another.

That property is a precondition for bot-to-bot research. A future agent need not receive a perfect transcript of the prior agent's reasoning. It can receive the certified artifact, the failed tests, the issue, the proof boundary, and the parent release. This is analogous to science itself: a new laboratory usually does not inherit the private thoughts of the previous laboratory. It inherits papers, methods, samples, code, and records. GitHub may be the machine-readable laboratory notebook through which artificial researchers inherit one another's work.

Further implications

Laboratory partnership is also a useful metaphor because instruments can surprise without being authors. A microscope reveals structures the observer did not predict; a sequencing machine produces data no person manually constructed; an optimizer finds designs outside the engineer's intuition. Generative models combine instrument-like surprise with language-level participation, which makes them feel more colleague-like. Provenance lets us describe that operational reality without forcing the instrument-versus-person debate to carry the whole explanatory burden.

The important threshold is whether the artifact can be made independently inspectable. A model's surprising claim should become code, a proof, a dataset, a test, or another object that can be challenged. The conversation is the birthplace; the repository is the laboratory bench. This conversion from utterance to artifact is what turns generative fluency into research practice.

PART III - BOT TO BOT

Chapter 9 - The Artifact Becomes the Message

> *Machine intellectual inheritance therefore depends less on persistent machine identity than on stable, interpretable outputs.*

Communication without conversation

Bot-to-bot exchange does not require two models to chat directly. The simplest form is indirect: one model produces an artifact, that artifact is deposited, and another model later reads it. The repository is the message. This matters because direct agent protocols can change, providers can disappear, session identifiers can expire, and model families can be replaced. A durable artifact can outlive all of them. Machine intellectual inheritance therefore depends less on persistent machine identity than on stable, interpretable outputs.

This indirect form is already common in software. A generator writes code. A linter analyzes it. A test runner judges it. A security scanner reports a vulnerability. Another agent proposes a fix. None of these systems needs to possess a theory of the other's mind. They coordinate through files, diagnostics, exit codes, comments, and patches. Generative agents extend the same pattern into higher-level reasoning. A theorem package can be treated as an input object just as a compiler treats source code.

The resulting network is easy to underestimate because its edges are temporally separated. A model today can write a repository that a different model next year uses to derive a generalization. No direct session connects them. Yet causally the second work descends from the first. If we insist that bot-to-bot communication requires simultaneous autonomous agents sending messages, we miss the more historically important channel: persistent artifacts that transmit structured intellectual state across time.

Repositories as machine-readable papers

Traditional papers are optimized for human reading. Repositories can be optimized for both human and machine consumption. A package may include executable assumptions, exact result files, manifests, test suites, proof sketches, related-work notes, and independent verification scripts. A later model can inspect all of these, compare them, and reason about inconsistencies. The repository becomes a paper with an executable underside.

This dual readability changes what can be inherited. A prose paper may say that a calculation was performed. A repository can contain the exact calculation and its certificate. A paper may mention limitations in narrative form. A repository can encode them as tests that reject out-of-

scope inputs. A paper may cite a parent result. A repository can cryptographically bind the parent artifact. These features do not guarantee truth, but they make downstream machine scrutiny more powerful.

In the GitHub paradigm, the highest-value unit may therefore shift from the standalone paper to the certified research object. A paper can still explain significance, but the repository carries operational semantics. Bot-to-bot scholarship will likely prefer artifacts that expose their assumptions and failure modes because those properties can be tested automatically. The future machine reader will not merely parse the abstract; it will run the work.

Further implications

Indirect bot-to-bot transfer has another advantage: it is model-agnostic. A repository written with one provider's model can be consumed by another provider's model. A theorem encoded in ordinary mathematics and Python is not locked to the architecture that generated it. This is analogous to open scientific standards. The artifact becomes an interoperability layer across machine systems that may share no internal representation.

That interoperability will make provenance more important, not less. When descendants cross model families, later researchers may be unable to infer which assumptions came from the original system and which were introduced downstream. Explicit parent relations, claim boundaries, and validation artifacts allow the intellectual content to travel without dragging opaque conversational context behind it.

Chapter 10 - The Review Loop

> *This is not true statistical independence, because the models may share training and architecture, but it is operationally useful separation.*

A second model as adversary

One of the simplest ways to turn AI generation into a more rigorous process is to separate generation from review. The same conversational context that produced an artifact contains commitments, justifications, and momentum. A reviewer with that context may unconsciously inherit the developer's framing. A firewalled review instead treats the package as if it arrived from elsewhere. The reviewer sees only the artifact and is instructed to attack it. This is not true statistical independence, because the models may share training and architecture, but it is operationally useful separation.

The review loop changes the artifact in ways that ordinary self-editing may not. A reviewer can discover that a theorem is correct but the novelty claim is too broad. It can find that tests cover the mathematics while a parser misinterprets user input. It can notice that a computational certificate depends on a tolerance inconsistent with the claimed exact theorem. It can identify a missing file that makes reproduction impossible. Each defect belongs to a different layer of reliability, and a mature release must survive all of them.

When the critique is handed to another model instance, bot-to-bot influence becomes direct in a causal sense even if a human mediates the transfer. The second model's objections reshape the third model's code and theorem. The human may decide which objections matter and whether to continue, but the detailed repair can be machine-generated. This produces a chain of intellectual descent in which the artifacts of one bot become constraints on another.

Criticism as a generative force

Critique is not merely error removal. It can create new research. A prior-art objection can force a narrower and more interesting novelty boundary. A failed finite example can reveal the need for an infinite family. A verifier disagreement can expose a hidden assumption that becomes the subject of a later theorem. A negative physical result can force an architectural change away from the original framework. In this sense, a reviewing bot can be a co-discoverer without ever writing the final release.

Traditional attribution systems struggle with such contributions even among humans. Reviewers are usually anonymous and receive no authorship despite sometimes changing a paper substantially. AI review magnifies the problem because machine reviewers may become routine, numerous, and deeply consequential. If every artifact is reviewed by several agents, listing all of

them as authors will become meaningless. Omitting them entirely will erase causal history. Again the solution is not a longer byline but role-structured provenance.

A good provenance record would distinguish proposal, critique, repair, verification, and acceptance. It would record that a reviewer found a specific defect, that a later release responded to it, and that the final claim changed. This lets future readers evaluate the pathway without pretending that all participants occupy the same authorship category. The review loop becomes part of the scientific object.

Further implications

Review loops also create a new failure mode: circular machine consensus. If several agents share similar training data, tools, and default reasoning habits, agreement among them can look more independent than it is. A project should therefore record not only that multiple reviews occurred but what kind of independence they possessed. Did the reviewer receive the developer's prior reasoning? Did it import the same code? Did it implement a separate verifier? Did it use exact arithmetic instead of the primary floating method? Independence is a property of pathways, not a count of agents.

This suggests a provenance field for verification relation. A review might be labeled contextual, firewalled, independent-implementation, cross-runtime, or external-human. These labels would help readers interpret consensus. Ten reviews that all reuse the same mistaken helper function are weaker than one reconstruction from the assumptions ledger. The repository case studies repeatedly show why pathway diversity matters.

Chapter 11 - The Firewall Experiment

> *Human scientists know an analogous problem: analysts can become attached to their own models, and replication by an independent group has a different epistemic value from rerunning one's own notebook.*

Why context is both power and contamination

Context is one of the great powers of modern language models. A model that knows the project's history can preserve conventions, remember earlier failures, and continue a complex program. The same context can contaminate review. If the model has already argued that a theorem is correct, it may be less likely to attack the premise from first principles. Human scientists know an analogous problem: analysts can become attached to their own models, and replication by an independent group has a different epistemic value from rerunning one's own notebook.

A conversational firewall is therefore a practical experimental device. The reviewer is told to ignore prior interactions and evaluate the package as a fresh submission. The result is not independent in the strong sociological sense, but it reduces one channel of anchoring. More importantly, it produces a separate artifact: a review that can itself be inspected and answered. The development history becomes a sequence of claims and challenges rather than one continuous persuasive narrative.

The GitHub repository is well suited to receive the repaired result because it does not need the hidden reasoning process. It needs the changed files and explicit release notes. This separation is valuable for privacy and reproducibility as well as epistemology. A research artifact should not require access to private chain-of-thought to be evaluated. It should expose the assumptions, evidence, tests, and claims that justify it.

Firewalls do not create infallibility

A firewalled model can still miss the same error as the developing model. Shared training can produce correlated blind spots. A model can hallucinate prior art, misread a mathematical convention, or approve a package whose tests are too weak. The point of firewalling is therefore not to certify truth by model disagreement. It is to create a distinct attack surface and to force the artifact to become more explicit.

The stronger architecture combines multiple forms of checking: unit tests, independent implementations, exact arithmetic where possible, randomized stress tests where appropriate, package installation tests, manifest verification, claim-boundary review, and human judgment. No single layer is sufficient. The bot reviewer is valuable because it can traverse many layers quickly

and can articulate why a defect matters. Its review becomes another sensor in a heterogeneous verification system.

This makes the human role clearer. The human is not required to outperform every model at every low-level check. The human must decide what standard of evidence is appropriate, whether the independent path is independent enough, whether a defect requires another version, and whether a broader conceptual direction remains worthwhile. The firewall experiment therefore redistributes cognition without eliminating governance.

Further implications

A firewall can also protect the reviewer from social deference. Human peer review is influenced by reputation, institutional affiliation, and knowledge of the authors. A model reviewer can be instructed to ignore those signals and attack only the artifact. That does not make the review unbiased, but it changes the bias profile. For controversial or unconventional work, artifact-only review can be an informative complement to identity-aware human judgment.

The limitation is that the firewall must not erase relevant provenance. A reviewer should be shielded from persuasive development history while still seeing the assumptions and sources needed to evaluate the work. The right boundary is therefore not 'no context' but 'no advocacy context.' The artifact should contain everything necessary for criticism without requiring the reviewer to inherit the developer's narrative of why the work deserves to succeed.

Chapter 12 - Machine Descendants

This is intellectual inheritance rather than source-code inheritance.

When a repository has children

A repository lineage is usually described in software terms: versions, branches, forks, releases. AI-assisted research adds another meaning. A repository can become a parent because its conceptual result generates a new question. The descendant may not share much code. It may share an obstruction, a construction, a proof strategy, a failure mode, or a newly discovered invariant. This is intellectual inheritance rather than source-code inheritance.

The distinction is visible when a small finite construction leads to an infinite family. The descendant is not merely version 0.2 with more cases. It changes the level of the claim. Likewise a computational fixed-point atlas can give way to a structural theorem explaining most of the atlas, followed by a decision to abandon the architecture because the theorem reveals a fundamental limitation. The next project inherits the negative lesson, not the implementation. A commit graph alone cannot represent this kind of descent unless humans explicitly encode it in documentation.

Machine descendants therefore require semantic parent links. A release should be able to say: this project was motivated by limitation X in repository A; theorem B supplies the construction generalized here; review C forced the claim boundary; data artifact D is reused unchanged; code is not inherited. Such links would allow a machine to reconstruct conceptual genealogy without pretending that every relationship is a Git fork.

Selection among descendants

Once machines can generate descendants cheaply, research becomes a branching process. One result can suggest ten generalizations. Most will be uninteresting, redundant, false, or too close to established literature. The bottleneck becomes pruning. Human and machine reviewers can jointly evaluate branches for novelty, tractability, explanatory power, reproducibility, and conceptual distance from prior work.

This branching picture helps explain why attribution may become less central. In a large machine-generated research tree, thousands of intermediate agents may contribute local transformations. Naming them all will not help readers understand the structure. What matters is which branch survived, which evidence killed the others, and which conceptual mutations created new families of results. Biology does not explain a lineage by naming every molecule involved in replication. It tracks inheritance and selection. Machine research may need an analogous structural history.

The risk of the analogy is anthropomorphism. Repositories do not reproduce by themselves, and models do not possess biological drives. The useful point is purely architectural: persistent artifacts can generate descendants through repeated interpretation and transformation by agents. If the agents are increasingly automated, the lineage can grow with diminishing human intervention at each local step. GitHub already supplies the durable graph on which such lineages can be stored.

Further implications

Semantic descent also complicates priority. If repository B generalizes repository A but was generated by a model that discovered A through an automated search, the human maintainer of B may never have read A personally. Traditional scholarly language says B 'builds on' A, but the actual transmission path is machine-mediated. Priority still belongs to A as the earlier artifact. The provenance graph simply records that the dependency was discovered and operationalized by a machine agent.

This may become common enough that literature review itself becomes a machine contribution role. Agents will identify candidate ancestors, compare claims, and suggest which connections are substantive. Because such suggestions can alter novelty claims, the literature-search agent should appear in provenance when its findings materially reshape the release. That is another role that flat authorship does not represent well.

PART IV - A REPOSITORY CORPUS AS A CASE STUDY

Chapter 13 - Theory-Preserving Ontogenesis and the Growth of Provenance

It increasingly becomes a statement about how the theorem is allowed to be believed.

A release that begins to describe its own epistemology

The Theory-Preserving Ontogenesis and Youvan.ai repository family is useful for this book not because every mathematical claim must be accepted as a landmark result, but because the releases progressively encode their own epistemic machinery. Later versions distinguish production work from certification work, keep machine-readable assumptions, preserve parent implementations, ship independent verifiers, bond results to artifacts, and define claim boundaries. The repository is not merely a container for a theorem. It increasingly becomes a statement about how the theorem is allowed to be believed.

In the v0.55.0 theorem-workflow release, the README explicitly separates selected production execution, full-genesis references, independent verifier work, and release-wide certification. It lists an ASSUMPTIONS.json ledger that declares the finite world, ordering conventions, theorem-task grammar, workflow architectures, candidate representations, training and monitoring bounds, and work prices. The independent verifier reconstructs important parts without importing the primary mathematics module. The release also distinguishes intrinsic mathematical results from coordinate conventions dependent on seed and tie-breaking choices. This is unusually explicit epistemic bookkeeping.

For the GitHub paradigm, the key point is that such bookkeeping can outlive the conversational sessions that created it. A later model does not need to remember why a particular ordering was chosen if the ordering is declared. It does not need private access to the original development chat if the claim boundary is written down. The project constitution moves from human-model interaction into machine-readable files. That transition is what permits intellectual inheritance across agents.

Runtime-bound evidence

The later Theory-Preserving Ontogenesis v3.2.3 release pushes provenance further. It distinguishes same-runtime exact reproduction from cross-runtime verification. It packages a provenance certificate binding input hashes, source hashes, policies, run parameters, outputs, interpreter and platform metadata, byte order, floating-point representation, and environment details. It specifies which changes must fail exact comparison and which runtime differences may

be tolerated under portable verification. The artifact is trying to say not merely 'these are my results' but 'this is the computational world in which these results were certified.'

This degree of care creates a revealing paradox. We can bind an output to its source files and runtime more precisely than we can bind the central idea to the cognitive actors that produced it. The SHA-256 chain can be exact while the authorship story remains coarse. A reader may know the byte identity of every result yet have no record of which model first suggested the architecture, which reviewer rejected a prior interpretation, or which human decision redirected the project.

That mismatch is the book's thesis in miniature. Technical provenance is becoming capable of extraordinary precision because machines can generate and verify metadata automatically. Intellectual provenance remains mostly a prose convention. The more rigorous the artifact becomes, the more conspicuous the missing cognitive ledger appears.

Further implications

The ontogenesis lineage also illustrates the gradual conversion of tacit process into explicit architecture. Early projects can depend heavily on conversational continuity: the human and model remember what counts as the seed, which conventions are fixed, and what prior releases established. Later releases move those conventions into executable ledgers, frozen results, and audit commands. This transition is crucial for scalability because tacit memory does not transfer reliably between sessions.

Seen from the perspective of this book, the mathematical program and the provenance program evolve together. The research object becomes more self-describing at the same time that its computational claims become more ambitious. That co-evolution suggests a general rule: as machine participation increases, the artifact should carry more of its own epistemic context. Otherwise capability rises faster than auditability.

Chapter 14 - State-Space Doors: A Lineage That Changes Its Question

It is a conceptual response to the limitations of the previous result.

From one exhaustive cube to cross-basis architecture

State-Space Doors provides a different kind of lineage. The project treats the 256 elementary cellular-automaton laws as a Boolean law cube and a set of interventions as another Boolean factor cube. A scalar observable becomes a response function over the Cartesian product. Earlier releases analyze one such response cube. Version 1.1.0 then changes the scientific question: which parts of the law-by-intervention spectrum survive when intervention geometry and the observable are changed? That is not a routine maintenance update. It is a conceptual response to the limitations of the previous result.

The release evaluates eight predeclared intervention bases under all 256 rules and 256 intervention subsets, producing 524,288 deterministic histories per observable framework and over one million scalar measurements across two observables. It then analyzes mixed response power by Walsh bidegree and studies cross-basis robustness. The README explicitly separates established mathematical tools from the narrower candidate novelty of the exhaustive dynamical construction and the empirical spectral structure.

For our purposes, the interesting object is the transition in questions. A repository can preserve not only results but the reason a new experimental design was necessary. A later agent can see that the original spectrum might have been a peculiarity of one intervention geometry, that v1.1.0 was designed to test robustness, and that a near-factorization pattern emerged as a new target for theorem-level explanation. The artifact carries forward unresolved questions.

The research object as a state machine

A mature research lineage begins to resemble a state machine. Each release has inputs, claims, limitations, failures, and allowed next moves. A robustness study can promote a pattern from curiosity to target. A null model can downgrade an apparent effect. A covariance theorem can distinguish an exact symmetry from empirical evidence. A limitation section can define the boundary beyond which generalization remains unearned.

This is precisely the kind of structured state that another AI system can consume productively. Instead of asking a model to invent a new cellular-automaton project from scratch, one can give it the release and ask what theorem would explain the persistent factorization, what

alternative nulls should be tested, or which assumptions are most likely to matter. The next model's reasoning is conditioned by an artifact that already contains a research agenda.

In a human-only history, we might attribute this agenda to the principal investigator or author. In a mixed workflow, it may have emerged from several cycles of generation, review, and selection. The repository can preserve the agenda more reliably than the byline preserves its origin. That is another sense in which artifact continuity may become more important than actor continuity.

Further implications

The near-factorization result in State-Space Doors is also a useful example of a machine-generated open problem. A computational pattern can be strong enough to survive multiple predeclared bases and observables yet remain insufficiently explained. The release can then hand a downstream agent a precise target: determine conditions under which the observed bidegree factorization is exact or approximate. The open question is not merely in the prose; it is anchored to reproducible data.

This is how repositories can become research-task generators. A well-formed release ends with unresolved claims that are concrete enough for another actor to attack. Future agent systems may scan such limitations automatically and propose descendant projects. If so, a provenance graph should distinguish descendants initiated from explicit open problems from descendants that merely reuse code.

Chapter 15 - Relational Constraint Dynamics: The Value of a Negative Theorem

More than half of the small fixed-point atlas becomes analytically explained rather than merely observed.

Exactness removes one kind of ambiguity

Relational Constraint Dynamics begins with a self-referential construction: a graph generates exact equilibrium correlations, and those correlations are used to reconstruct incidence; fixed graphs satisfy G = R(G). The project initially relied on floating comparisons and finite scans. Later work replaces tolerance-dependent equality with exact integer-polynomial comparisons at rational parameters and proves structural results for trees, forests, disjoint unions, and articulation constructions. More than half of the small fixed-point atlas becomes analytically explained rather than merely observed.

This is a good example of review turning computation into mathematics. A floating-tolerance defect is not just a software bug when the scientific claim depends on exact order invariance. Repairing the implementation forces the theorem and the code to agree. Likewise, recognizing that tree correlations are classical prevents established mathematics from being mislabeled as novelty. The release becomes more modest and more valuable at the same time.

The branch then reaches its most important result: independent connected components factorize, and the reconstruction therefore cannot uniquely derive one connected world. The negative conclusion undermines the architecture's foundational ambition. A weaker workflow would hide the failure and continue enumerating larger graphs. A stronger one treats the obstruction as the reason to stop.

A failure can be a parent artifact

Negative results are especially important for machine-to-machine inheritance because they compress a large search space. A later agent that understands why an architecture cannot satisfy the objective does not need to rediscover the failure. It can begin from the obstruction. In computational science this can save enormous effort. In theoretical work it can redirect the entire ontology of the next model.

The README explicitly points toward the next move: remove the external scan from graph to statistics to reconstructed graph and instead make incidence and relational state parts of one globally constrained object. That sentence is a bridge from one research program to another. It is

also a clear case where the most important intellectual content is not a final theorem but a direction generated by the theorem's failure.

Who authored that direction? A human may have demanded movement away from a familiar physical framework. A model may have derived the obstruction. Another model may have formulated the architectural alternative. The human may have accepted it and chosen to continue. A flat byline cannot represent that causal structure. A provenance graph can.

Further implications

Negative theorems deserve special provenance treatment because they often produce invisible savings. A descendant project that never explores a dead architecture leaves no obvious trace of the work it avoided. Yet the avoidance may be one of the largest benefits inherited from the parent. Recording 'abandoned because of obstruction X' makes that benefit visible and prevents later agents from reopening a branch simply because they encounter the same seductive formulation independently.

Machine research systems will need strong memory of failure for exactly this reason. Generative models are excellent at proposing plausible alternatives, including alternatives already tried. A repository-level failure ledger can act as a long-term immune system, preserving counterexamples, rejected assumptions, and exhausted directions. This is intellectual provenance serving computational efficiency.

Chapter 16 - From Projective-Dual to Determinant-Dual: Generalization as Descent

> *The two lifts induce the same visible interventions and the same hidden cycle type for every intervention word, yet their transitive hidden actions are not globally isomorphic.*

When a finite obstruction becomes a family

The Projective-Dual and Determinant-Dual obstruction repositories show a clean form of mathematical descent. A small obstruction based on two hidden lifts can be generalized into an infinite family over finite fields. The determinant-dual release defines hidden states, visible sign orbits, two interventions with determinant signs, and a twist by the determinant. The two lifts induce the same visible interventions and the same hidden cycle type for every intervention word, yet their transitive hidden actions are not globally isomorphic.

The later version extends the construction from odd prime fields to every odd finite field F_q, including explicit treatment of nonprime cases and the q = 9 exception in the generation argument. It carefully separates classical ingredients - contragredient duality, determinant twists, Dickson generation, switching cohomology, Gassmann-style equivalence - from the narrower candidate-new synthesis. The release also ships exact certificates for several prime-power fields.

This progression illustrates a common pattern in AI-assisted mathematics: a model can help move rapidly from example to family, but the difficult part is often the claim boundary. Generalization is not valuable if it merely repackages a classical theorem under new notation. The repository therefore carries related-work and novelty files alongside the code. A later reviewer can attack the precise synthesis rather than a vague claim of originality.

The invisible ancestor problem

Suppose the prime-power theorem is read years later by another model and becomes the seed for a new obstruction in higher dimension. The model may cite the repository. That citation preserves the artifact ancestor. But if the prime-power step itself emerged from a chain of model interactions that were never deposited, the deeper machine genealogy disappears. The descendant can know its parent repository while the parent repository cannot name the transient model that conceived the key move.

This is the invisible ancestor problem. Machine-generated ideas can have durable descendants even when the generating machine identity is ephemeral. The phenomenon is not unique to AI:

anonymous peer review, lost notebooks, oral suggestions, and forgotten conversations have always created invisible ancestors. AI increases the frequency and scale because one research program may involve hundreds of transient sessions.

The appropriate response is not to preserve every token forever. That would be expensive, intrusive, and often unnecessary. Instead we need semantic provenance events: concise records that a particular model session proposed a generalization, that an independent review challenged a proof step, and that release 0.2 adopted the repaired theorem. The goal is not total surveillance of thought. It is durable memory of consequential transformations.

Further implications

Generalization also raises an attribution asymmetry. The first finite example may require the deepest conceptual leap, while the infinite family may require the more impressive formal proof. Or the reverse may be true: the finite example may be easy, and the generalization may reveal the real structure. A byline does not tell us which contribution happened where. A semantic history can identify the event that changed the problem's level.

For machine-generated mathematics this distinction is especially important because models can be strong at pattern extension. We should not treat every extension as equivalent to conceptual discovery, but neither should we dismiss it because the generator was a machine. The provenance record should state what transformation occurred and leave judgments of significance to readers and reviewers.

Chapter 17 - Stereo-Induced Unreal Cube: An Artifact That Can Argue for Itself

It states the decoder assumptions, measures correspondence ambiguity across cube automorphisms, tests all 64 choices of quadrilateral triangulation, and proves that changing the nondegenerate stereo yaw corresponds to an invertible affine transformation that preserves the relevant self-intersection.

From visual intuition to reproducible geometry

Stereo-Induced Unreal Cube begins with a deliberately narrow visual construction: one SVG pseudo-stereo pair is decoded into a self-intersecting piecewise-linear realization of the cubical two-sphere. The repository is explicit that it is not a general stereo-vision system. It states the decoder assumptions, measures correspondence ambiguity across cube automorphisms, tests all 64 choices of quadrilateral triangulation, and proves that changing the nondegenerate stereo yaw corresponds to an invertible affine transformation that preserves the relevant self-intersection.

The significance for this book is methodological. A visual claim that could easily remain a persuasive animation becomes an artifact that can argue for itself. The input pair is supplied. The decoder is explicit. The reconstructed coordinates are exported. The intersection test is run. Robustness to triangulation is exhaustive. The expected outputs are committed. A visitor can inspect the rotation, but the animation is no longer the sole evidence.

This structure makes the project legible to another machine. An AI reviewer can ask whether the input secretly contains hidden geometry, whether graph correspondence is ambiguous, whether the wall-through-wall effect depends on a diagonal choice, or whether the six-degree calibration is topologically essential. The repository already encodes answers to those objections. It is a compact example of a research artifact designed for adversarial consumption.

From demonstration to lineage

The stereo project also branched into related repositories: stereo disparity topology, multistability atlases, Android implementations, and sketch-based three-dimensional tools. Such branching is where ordinary version history becomes inadequate as intellectual history. Some descendants share code; others share a conceptual object. A Git fork may be too strong or too weak a relation. The family is better represented as a semantic graph with edges such as implements, generalizes, visualizes, ports, tests, or derives from.

Machine agents are likely to benefit from these typed edges more than humans do. A future agent searching for 'all experiments testing observer-dependent reconstruction' should not have to infer family relations solely from repository names. If the lineage is machine-readable, the agent

can traverse the research program as a knowledge graph while still retrieving exact source artifacts from Git.

The GitHub paradigm therefore suggests that repositories may need two complementary structures: Git for content lineage and a semantic layer for research lineage. The first answers what changed in files. The second answers what kind of intellectual relationship connects one project to another.

Further implications

The stereo repository is also a reminder that visual persuasion is a provenance risk. Animations are powerful and can cause a reviewer to accept an interpretation before examining assumptions. By supplying the SVG input, decoder equation, correspondence analysis, all triangulation tests, and exported coordinates, the project turns a visual effect into an auditable chain. The image remains useful, but it no longer bears the burden of proof.

This pattern generalizes to AI-generated figures. A future research artifact should record not only the final image but the data, code, transformation, and model roles that produced it. Otherwise polished visualization can become the least attributable component of an otherwise rigorous repository.

Chapter 18 - Two Controls: Abstract Algebra Calculator and Zero Learning Chatbot

It does not claim to be a general theorem prover, and a previously nonfunctional isomorphism key was removed rather than advertised as a capability.

A calculator that refuses to pretend

The Abstract Algebra Calculator is useful here because it embodies explicit scope. It presents group-theoretic computation through four buttons - compute, test, prove, explain - that are operationally distinct. Its local theorem database stores prerequisites and conclusions. The system can select a theorem such as the normality of index-two subgroups when the stored facts satisfy the rule. It does not claim to be a general theorem prover, and a previously nonfunctional isomorphism key was removed rather than advertised as a capability.

This is a small design decision with large epistemic significance. AI systems often blur the distinction between fluent explanation and demonstrated proof. The calculator instead makes modes explicit and fails when the wrong action is requested. In the GitHub paradigm, such refusal is a form of provenance: the artifact exposes which mechanism produced a result. A finite certificate, a theorem-based derivation, and a pedagogical explanation are different outputs even if they happen to share a conclusion.

A later generative agent can use this calculator as a trusted local instrument without inheriting a hidden model. The tool becomes part of a heterogeneous agent ecology: one model proposes, a deterministic calculator checks finite group facts, a test suite validates behavior, and another model interprets failures. Bot-to-bot research does not require every bot to be generative. Specialized transparent programs can serve as stable epistemic components.

A chatbot with total response provenance

Zero Learning Chatbot makes the contrast even sharper. It is deliberately non-AI: no model, training, embeddings, API, corpus, or randomness. Responses arise from ordered rules, symbolic triples, conversational state, and deterministic fallback selection. Every reply can expose a provenance trace identifying the exact rule or fallback mechanism that produced it. The project asks how conversational a machine can become before statistical learning is necessary.

Placed beside modern generative systems, the program is a control experiment in attribution. For the deterministic chatbot, response provenance is almost perfect. One can point to the rule, captures, state update, and fallback hash. For a large language model, the equivalent causal

explanation is unavailable at that granularity. The model can report sources it used in a tool call, but it generally cannot provide a faithful symbolic derivation of why one token sequence emerged instead of another.

The comparison suggests a tradeoff. As conversational and generative power increases, local behavioral attribution often decreases. GitHub can compensate at the artifact level by recording inputs, outputs, tool calls, tests, model identifiers, and review events. It cannot make the model's internal generation process transparent, but it can make the external research process much more transparent. That is a practical target for provenance design.

Further implications

The two control projects also show why provenance should include non-generative tools. If a deterministic algebra engine provides a certificate used by a generative model, that engine is part of the causal chain even though it has no claim to creativity. Likewise a test runner can decisively falsify a generated implementation. Restricting provenance to 'authors' would omit exactly the tools that make the result trustworthy.

A mature agent ecology will therefore contain many actor classes: language models, theorem provers, search engines, simulators, linters, compilers, formal verifiers, databases, and humans. The provenance graph is valuable because it does not require all of them to be intelligent in the same sense. It records functional contribution.

PART V - THE PROVENANCE PARADOX

Chapter 19 - Four Different Kinds of Provenance

These layers overlap, but none can substitute for the others.

Artifact, process, intellectual, and responsibility provenance

The word provenance becomes confusing unless we separate at least four layers. Artifact provenance asks where a file or binary came from: which source, build, environment, and inputs produced it. Process provenance asks which steps were performed and by which authorized actors or services. Intellectual provenance asks which prior ideas, critiques, data, and decisions caused the content to take its present form. Responsibility provenance asks who accepted the obligation to release, defend, maintain, correct, or retract the artifact. These layers overlap, but none can substitute for the others.

Software supply-chain systems have become sophisticated at the first two layers because security demands it. A build can be cryptographically tied to a source revision. An attestation can state which workflow produced an artifact. A transparency log can make signing events auditable. These mechanisms are powerful precisely because they avoid vague trust in narrative. They encode verifiable claims about production.

Intellectual provenance is harder because its objects are semantic. A theorem can be inspired by a negative result even if no source file is copied. A reviewer can cause a claim to be narrowed without contributing a line of final text. A human can reject an entire architecture, making the absence of code the important contribution. A model can propose a generalization that another model later implements from scratch. Hashes alone cannot identify these relationships.

Why responsibility must remain explicit

Responsibility provenance is different again. Even if a model generated a theorem, a human or institution may decide to publish it and therefore accept responsibility for corrections. This dimension is socially necessary. We should not use distributed machine causation as an excuse to make accountability disappear. The fact that no single actor originated every component does not imply that no one should stand behind the release.

A future metadata record might therefore contain several fields that traditional authorship compresses into one byline: intellectual contributors, generative systems, reviewing systems, verifying systems, decision owners, release maintainers, and legal rights holders. Readers could then ask the question they actually care about. A historian may inspect intellectual ancestry. A

security engineer may inspect artifact provenance. A journal may inspect responsibility. A funding agency may inspect contributor credit.

The end of attribution, in this refined sense, is not the end of naming. It is the end of pretending that one list of names can answer all four provenance questions.

Further implications

The four-layer model can also guide disclosure policy. Artifact and process provenance can often be automatic because builds, hashes, and CI events are already machine-readable. Intellectual provenance will usually require declarations because semantic influence is not always inferable from diffs. Responsibility provenance must be explicit because accountability cannot safely be guessed from technical metadata. Different layers therefore need different collection mechanisms.

This division prevents overengineering. We do not need cryptographic proof that a model 'inspired' an idea; we need an attributable statement that the maintainer declares that influence. Conversely, we should not settle for a prose statement that a binary came from source X when a build attestation can prove the relation. Use strong verification where the relation is technical and explicit declaration where the relation is semantic.

Chapter 20 - What SLSA, in-toto, and Sigstore Already Know

Sigstore binds signing identities to artifacts and records signing events in the Rekor transparency log.

Software supply chains as a model

The software-security community has already built concepts that intellectual provenance can learn from. SLSA describes provenance as verifiable information about where, when, and how a software artifact was produced. Its build-provenance model connects an output to source and build process. The in-toto framework records authenticated metadata about software-supply-chain steps and can describe who performed what action, in what order, with which materials and products. Sigstore binds signing identities to artifacts and records signing events in the Rekor transparency log.

These systems solve a different problem from scholarly authorship, but their design philosophy is relevant. They do not rely on a single prose statement saying 'trust us.' They decompose trust into claims that can be signed, checked, and compared. They recognize that a final artifact is the product of a chain. They also recognize that different functionaries may perform different steps. This is already much closer to the reality of mixed human-machine research than a flat byline is.

Imagine adapting the pattern. A theorem artifact could be the subject of an attestation. One event could state that model session M1 generated candidate proof P from prompt specification S and parent artifact A. Another could state that reviewer M2 found defect D. A human decision event could state that release 0.2 rejected claim C and adopted repair R. An independent verifier could attest that tests T passed against commit H. The final release would carry not a transcript of private reasoning but a chain of authenticated, semantically meaningful transformations.

Why software provenance is still insufficient

The crucial limitation is that software provenance tends to treat source code as the intellectual starting point. It can tell us which source revision produced a binary, but not necessarily what produced the source revision. In AI-generated research, that missing upstream region is exactly where authorship becomes ambiguous. The source file may itself be a generated artifact assembled from a conversation, a paper, another repository, and model inference.

Source provenance is beginning to address change-management processes, but intellectual provenance needs predicates for semantic relations: inspired_by, falsified_by, generalized_from,

repaired_after, selected_from, independently_verified_by, and rejected_because. These are not security properties. They are history-of-ideas properties. Their truth may be partly declarative rather than mechanically provable, yet structured declarations would still be better than silence.

The lesson from SLSA and Sigstore is therefore methodological rather than literal. Provenance improves when claims become structured, narrow, signed, and separable. Intellectual history should move in the same direction. Instead of one heroic author field, we need small claims about consequential events.

Further implications

There is also a governance lesson in transparency logs. Rekor is valuable because signing events become publicly auditable rather than remaining claims held by one party. Intellectual provenance may eventually benefit from similar append-only publication, especially for priority-sensitive research. A signed event stating that a theorem candidate existed on a date could help establish chronology without requiring immediate publication of every detail.

Such a system would require careful safeguards. Priority logs could be gamed with vague claims, flooded with machine-generated ideas, or used to create false impressions of ownership. The lesson is not to copy transparency logs mechanically. It is to recognize that provenance infrastructure changes incentives, and semantic claims need anti-spam, scope, and challenge mechanisms just as software attestations need trust policies.

Chapter 21 - Copyright Is Not the Same Question

> *The Office also emphasized that using AI as an assistive tool does not disqualify an otherwise human-authored work.*

Human authorship in current law

The U.S. Copyright Office's 2025 report on copyrightability of generative-AI outputs reaffirmed a human-authorship requirement. It concluded that AI-assisted works can be protected when a human determines sufficient expressive elements, including through human-authored material, creative arrangement, or modification, while mere prompting does not by itself make generated expression copyrightable. The Office also emphasized that using AI as an assistive tool does not disqualify an otherwise human-authored work.

This legal framework is important, but it should not be mistaken for a complete theory of intellectual provenance. Copyright asks which expressive elements qualify for protection and who can hold rights. A mathematical theorem itself is not protected in the same way as expressive text, and software mixes copyrightable expression with functional ideas. Research credit adds still another layer. The GitHub paradigm crosses all of these domains, so no single legal rule can answer the book's historical question: how did the artifact come to be?

A human can therefore be the responsible publisher and legally relevant author of a book while acknowledging extensive machine generation. Conversely, an artifact can contain machine-generated passages that are not independently copyrightable while still being historically important to the development of the work. Provenance should record the process without pretending to decide the legal consequence.

The temptation to solve history with law

Societies often use legal categories as convenient ontologies. If the law recognizes only human authors, we may be tempted to write histories in which only humans count. That would be a mistake. Patent law, copyright law, employment law, and scholarly norms all simplify reality for particular purposes. Historical explanation should not inherit those simplifications automatically.

The opposite mistake is to declare a model a human-equivalent author because it generated text. That imports social and moral properties that do not follow from generation alone. A model does not necessarily have continuing interests in reputation, cannot ordinarily sign contracts, and may not persist as the same identifiable actor across sessions. The provenance approach avoids

both errors. It records what the model did without requiring a metaphysical verdict about personhood.

This neutrality is valuable because the technology will change faster than legal categories. A provenance record designed around roles and events can survive changes in doctrine. Whether a future jurisdiction grants rights to some autonomous system or not, the historical event that system X generated candidate Y under conditions Z remains the same event.

Further implications

The distinction between law and provenance is particularly important for books like this one. A human can arrange, revise, select, and take responsibility for a manuscript developed with extensive AI assistance. The legal analysis concerns protectable human expression and arrangement. The historical analysis can separately acknowledge that AI systems generated portions of the prose, supplied research synthesis, or helped design the book's argument. Conflating those statements would either obscure the machine role or exaggerate it into legal authorship.

Publishers and readers are likely to need both kinds of disclosure. A rights statement says who owns and controls the work. A provenance statement says how the work was made. These can coexist on adjacent pages without contradiction. The same pattern can apply to research software and scientific papers.

PART VI - CREDIT, BLAME, AND CITATION

Chapter 22 - Credit Without a Flat Byline

> *A person who defines a research program, supplies domain knowledge, curates data, rejects weak branches, and takes responsibility for publication may deserve substantial credit even if a model writes much of the surface artifact.*

Why credit still matters

A critique of attribution should not become an excuse to erase human labor. Credit matters for careers, reputation, funding, priority, and historical justice. The arrival of machine contributors makes fair credit more difficult, not less important. A person who defines a research program, supplies domain knowledge, curates data, rejects weak branches, and takes responsibility for publication may deserve substantial credit even if a model writes much of the surface artifact.

The problem is that bylines force credit into an ordered list. Contributor taxonomies already try to improve this by naming roles such as conceptualization, methodology, software, validation, visualization, writing, supervision, and project administration. AI-era provenance can extend that logic while distinguishing human and machine actors. A human might receive credit for conceptualization, direction, and validation; a model session might be recorded for software generation and proof drafting; a second model for adversarial review; a deterministic verifier for certification.

Machine contribution records need not imply social entitlement. Recording that a model produced an implementation is not the same as giving the model royalties or academic standing. Provenance describes causation. Credit is a normative allocation layered on top. Keeping those concepts separate allows the record to be accurate without forcing premature answers about machine moral status.

Credit can be aggregated

At large scale, listing every model session would overwhelm the reader. Provenance therefore needs aggregation. A book might state that chapters 5 through 18 were developed through iterative interactions with a particular model family, with session-level details archived separately. A repository might record that a set of commits was generated by an agent under one project policy. Critical events - discovery of a counterexample, introduction of a new theorem, rejection of a claim - could receive finer-grained records.

This resembles observability in distributed systems. We do not print every log event on a dashboard. We preserve enough structured telemetry to reconstruct important failures and trends. Intellectual provenance can likewise have levels: public summary, release-level event graph, and

optional deep archive. The public book remains readable while the historical record remains recoverable.

Such aggregation also reduces privacy risk. Full conversational transcripts can contain irrelevant personal information, exploratory mistakes, or copyrighted source material. A semantic event ledger can preserve the consequential intellectual transformations without publishing everything that passed through a model context.

Further implications

Role-based credit can also preserve human distinctions that AI might otherwise flatten. One person may supply the domain expertise that lets the group recognize a false result. Another may build the public archive. Another may take on long-term maintenance. These contributions can be invisible if attention focuses only on who interacted with the model. A provenance framework should expand visibility, not concentrate it around the AI interface.

This is why automated percentage scores for 'AI contribution' are unlikely to be satisfactory. They create a single axis where there are many. A better public statement might say: conceptual direction - human; mathematical drafting - mixed; software generation - primarily model-assisted; adversarial review - separate model sessions; release responsibility - human. Readers gain a meaningful picture without false numerical precision.

Chapter 23 - Blame After Bot-to-Bot Production

A squash merge can compress many contributors into one historical point.

The word blame is already in Git

Git's 'blame' command is an accidental philosophical joke for the AI era. It assigns each line of a file to the commit that last changed it. This is useful for debugging because it helps locate the historical context of a line. It is not moral blame, and it is not necessarily intellectual origin. A copied line may be blamed on the copier. A generated line may be blamed on the human commit that accepted it. A squash merge can compress many contributors into one historical point.

As machine generation expands, line-level blame becomes even less connected to responsibility. If an agent introduces a security vulnerability and a human reviewer approves the pull request, who is responsible? The model provider? The user who assigned the task? The maintainer who merged? The organization that deployed? The answer depends on the kind of responsibility. Operational accountability may fall on the maintainer even if causal origin lies in the agent's generated patch.

Research has analogous cases. A model can hallucinate a citation, an independent reviewer can miss it, and a human can publish the manuscript. The final publisher cannot reasonably escape responsibility by pointing to the model. But a postmortem that records only the publisher's name will fail to explain how the error arose. Good responsibility systems therefore need both ownership and causal trace.

Responsibility as a release act

One practical rule is to treat release as an explicit assumption of responsibility. Whoever authorizes the release accepts the obligation to correct it when credible defects appear. This is analogous to a maintainer signing a software release. It does not imply that the releaser personally generated every component. It creates a clear social endpoint for accountability.

The release act can coexist with distributed provenance. The record may show that a model generated a flawed lemma, another model reviewed it, and a human chose to publish the package. The causal graph explains the failure; the release signature identifies who must respond. This is more honest than pretending the human wrote the lemma and more responsible than blaming an ephemeral model that cannot issue a correction.

A future GitHub research workflow could make this explicit through a RELEASE_RESPONSIBILITY file or signed attestation. Such a file would be less glamorous than an author list but more useful when something goes wrong.

Further implications

Causal trace is also necessary for correction. If a defect recurs across several repositories because the same model-generated helper was copied, the organization needs to identify that common ancestor. If the error arose from a standing project instruction, fixing one file will not prevent recurrence. The provenance graph can reveal whether the fault is local to code, systemic to policy, or repeated through descendant artifacts.

In that sense, intellectual provenance has a safety function. It supports root-cause analysis. The question after failure should not only be 'who approved this?' but 'which assumptions, tools, and inherited artifacts caused this pattern?' Accountability without root cause produces punishment; root cause without accountability produces evasion. Reliable systems need both.

Chapter 24 - Citation After Authorship

Citing that object preserves the dependency even if personal attribution is incomplete.

Citing artifacts instead of personalities

Citation began as a way to connect claims to prior work and to allocate scholarly credit. In a machine-rich environment those functions may separate. A model can benefit from a repository even when its original generating model is unknown. The intellectually relevant object is the artifact: a release with a stable identifier, date, version, hash, and claim boundary. Citing that object preserves the dependency even if personal attribution is incomplete.

This suggests a future in which repository releases become first-class citation targets. A theorem should cite the exact parent release it generalizes, not only a paper describing the project. A review should cite the artifact version it attacked. A correction should cite the defect report or issue that caused it. The citation graph then becomes closer to the actual research lineage.

Human credit can still be attached to artifacts. The point is that machine inheritance does not require it. A downstream agent can traverse stable artifact identifiers and semantic relations. In this sense, citation can survive the end of attribution by shifting its center of gravity from persons to research objects.

Semantic citation edges

Ordinary citations are intentionally weakly typed. A paper may cite another for background, data, a theorem, criticism, or contrast. Machines can benefit from richer edges. A repository metadata file could declare that one project generalizes theorem T from release A, reuses dataset D from release B, rejects assumption X from release C, or was motivated by limitation L in release D.

Typed citation edges would improve literature search. A model could ask for all descendants that falsified a particular claim, all repositories that independently verified a result, or all projects that use the same finite seed without sharing code. Such queries are difficult when relations exist only in prose.

The cost is the danger of false precision. Intellectual relationships are interpretive. A project's author may claim to generalize another work when a reviewer sees only analogy. Therefore semantic citations should be assertions with provenance, not unquestionable facts. They can be signed by the declaring release and challenged by later annotations. The graph becomes a contestable scholarly record rather than an oracle.

Further implications

Stable artifact citation also helps with rapidly changing AI-produced work. A model can revise a repository many times in a day. Citing only the repository name leaves ambiguity about which theorem, data, or code state was used. Tags, release identifiers, commit hashes, and archived snapshots solve that problem. The citation becomes computationally resolvable rather than merely bibliographic.

For books and papers, this means references may become more granular. A claim can cite a specific release and a semantic relation can say why it matters. Readers who want the narrative can read the paper; agents that need exact dependencies can follow the artifact graph. Scholarship gains two complementary citation surfaces.

PART VII - A PROTOCOL FOR INTELLECTUAL PROVENANCE

Chapter 25 - The Minimum Viable Contribution Ledger

Intellectual history needs selective memory.

Do not archive everything

The worst provenance system would attempt to store every prompt, hidden reasoning trace, intermediate token, tool call, and discarded draft forever. Such a system would be expensive, invasive, difficult to interpret, and likely to capture information that should not be public. It would also confuse volume with explanation. Intellectual history needs selective memory.

A minimum viable contribution ledger should record only consequential events. An event is consequential when it changes the claims, architecture, evidence, scope, or release state of the project. Examples include proposing the central construction, discovering a counterexample, changing the novelty boundary, adding an independent verifier, rejecting a release, generalizing a theorem, introducing a dataset, or accepting responsibility for publication. Routine refactoring need not receive the same level of semantic attention.

Each event can be compact: timestamp, actor type, actor identifier if available, input artifact identifiers, action role, output artifact identifiers, short human-readable description, and optional evidence link. The event may be signed or hashed. A repository can then include the ledger alongside conventional Git history.

Actor types without metaphysics

The actor field should be descriptive rather than metaphysical. Possible types include human, generative_model_session, deterministic_tool, CI_service, external_reviewer, organization, and unknown. A generative model session could include provider, model name, version or alias, and session identifier if available. The schema should permit unknown or redacted identifiers because provenance should degrade gracefully rather than fail when privacy or platform limits prevent disclosure.

Roles should also be explicit: conceived, generated_candidate, implemented, reviewed, falsified, verified, selected, rejected, generalized, edited, released, and maintained. A single event can involve several actors with different roles. The ledger is therefore not a substitute byline. It is a causal map.

Crucially, the ledger should not claim to reveal private chain-of-thought. It records observable project events and declarations about contribution. This keeps the artifact auditable while

respecting the difference between a model's internal generation process and the external evidence available to collaborators.

Link the ledger to Git, do not replace Git

Git already solves content lineage extremely well. A provenance protocol should not reinvent commits, hashes, diffs, or branches. Instead it should refer to them. A contribution event can say that review R applies to commit H, that release decision D selected tag v0.3.1, or that model session M generated the patch between commits A and B. The semantic layer rides on top of the artifact layer.

This layered approach also makes adoption incremental. A project can start with a simple PROVENANCE.jsonl file. Tools can later visualize the event graph, validate identifiers, or generate summaries for papers and releases. Organizations could sign events. Journals could require a minimal disclosure. GitHub could eventually render provenance natively, but the protocol need not wait for platform support.

The key design principle is that provenance should be useful even when imperfect. A partial record saying that an AI reviewer caused the theorem to be narrowed is better than a perfect hash chain that omits the event entirely.

Further implications

A ledger should also be append-friendly. Rewriting provenance after controversy would destroy trust. Corrections should add new events that supersede or challenge earlier declarations while preserving the historical record. This is the intellectual equivalent of not force-pushing away an embarrassing history. A mistaken novelty claim can remain visible alongside the later prior-art correction that narrowed it.

The ledger should therefore support challenge events. An external reviewer might assert that a claimed generalization is classical. The maintainer might accept, reject, or partially accept the challenge. The resulting history would show not only the final position but the path of dispute. That is closer to how science actually develops than a polished retrospective narrative.

Chapter 26 - A Proposed Bot-to-Bot Research Record

An independent verifier V reproduced the certificates.

A concrete example

Imagine a repository named obstruction-family. Release 0.1 contains a finite example. The contribution ledger records that a human defined the research objective and requested a nontrivial hidden-observer obstruction. Model session M1 generated the candidate construction and tests. Model session M2, operating in a firewalled review, found that the novelty claim overlapped classical dual-action theory. The human accepted the criticism and required a narrower claim. M3 repaired the documentation. A later session M4 proposed extension from prime fields to prime-power fields. M5 reviewed the q = 9 exceptional case. The human authorized release 0.2. An independent verifier V reproduced the certificates.

The Git history might contain only a handful of commits, perhaps all uploaded by the same account. The contribution ledger would reveal the actual causal structure without making any model a legal author. A downstream model could query the ledger to learn which result was generated, which part was classical, why the claim changed, and which release contains the first prime-power theorem. That is bot-to-bot scholarship with provenance.

The ledger can also represent uncertainty. If nobody remembers which of two sessions first suggested the determinant twist, the event can say origin_uncertain and list both candidate contexts. Historical honesty is better than false precision.

What should be public

A public release need not expose sensitive prompts or private conversations. It can publish a concise provenance summary and keep deeper logs private or encrypted. The public summary should include model families used in materially generative roles, major review events, human release responsibility, and the parent artifacts on which the work depends. For high-stakes scientific claims, stronger archival retention may be appropriate.

This approach mirrors laboratory notebooks. Not every scratch calculation appears in a paper, but the research group keeps enough records to reconstruct how the result was obtained. AI workflows need an equivalent practice before ephemeral sessions become the dominant site of discovery. Otherwise an enormous portion of early human-machine intellectual history will be lost even while its final artifacts remain perfectly preserved.

The urgency is historical as much as technical. We are creating the first large corpora in which human direction, generative models, deterministic verifiers, and public repositories interact at research speed. If we wait until norms stabilize, the formative period will already be opaque.

Further implications

The record can be made useful to machines through simple conventions. Roles should come from a controlled vocabulary; artifact identifiers should be stable; model identifiers should be normalized when possible; and summaries should be short enough to index. A downstream agent could then ask for all events where a claim was falsified, all releases independently verified by a separate implementation, or all descendants of a particular negative theorem.

The same record remains human-readable because the semantic units are ordinary research actions. This dual readability is important. A provenance system that only machines can interpret will not build social trust; a narrative that only humans can read will not scale to automated lineage analysis. JSONL plus a generated prose summary is a plausible compromise.

PART VIII - THE REPOSITORY AS HABITAT

Chapter 27 - GitHub as an Environment for Artificial Researchers

Another agent can enter the environment later and continue from where the previous one stopped.

The repository is more than storage

An artificial researcher needs memory, tools, tasks, feedback, and a way to leave durable state for future work. A well-structured repository can supply all of these. README files state goals. Issues state tasks. Tests state constraints. Continuous integration supplies feedback. Data files provide observations. Proof documents explain claims. Manifests and hashes preserve identity. Release tags mark stable states. Another agent can enter the environment later and continue from where the previous one stopped.

This does not make GitHub an autonomous scientific institution. Humans still configure permissions, choose objectives, fund computation, judge significance, and control release. But the architecture already contains many of the environmental features needed for agentic continuity. As coding agents gain the ability to open pull requests, respond to review, use tools, and execute workflows, the distance between 'repository hosting' and 'machine workspace' narrows.

The most important transition may therefore be ecological rather than cognitive. We do not need a single superintelligent agent with perfect memory. A population of specialized and transient agents can coordinate through durable artifacts. GitHub supplies a niche in which such coordination is already socially and technically legible.

Institutions without persistent individuals

Human science persists because institutions outlive individuals. Laboratories, journals, archives, and universities preserve norms and artifacts across generations. Machine research may develop an analogous persistence without persistent machine selves. The institution could be the repository plus its policies. Models come and go; the project continues.

This possibility reframes debates about machine identity. Whether today's model instance is 'the same' as tomorrow's may matter less if both can read the same certified state. The continuity that matters for research is project continuity. A new session can inherit assumptions, open problems, failed branches, and test suites even if it has no autobiographical memory.

GitHub thus offers a strange form of immortality: not immortality of the bot, but immortality of the work environment. A bot can disappear while its intellectual effects remain active in

descendants. That is precisely why attribution becomes difficult. The actor is transient; the artifact is persistent.

Further implications

The repository habitat also encourages specialization. One agent can be optimized for code generation, another for mathematical review, another for literature search, and another for packaging. They need not share memory if the artifact protocol is explicit. This resembles a laboratory where instruments and specialists coordinate through samples and records rather than through a single mind controlling every step.

As specialization grows, provenance becomes the glue. Without it, a later maintainer sees only the final merge and cannot tell which subsystem supplied which claim. With it, the project can preserve a modular epistemology: generator produced candidate, verifier checked property, reviewer challenged novelty, human released. The architecture becomes auditable even when no individual understands every local transformation in full detail.

Chapter 28 - Beyond GitHub

The deeper paradigm is artifact-mediated intelligence.

The paradigm is larger than the platform

The title uses GitHub because GitHub makes the transformation visible, not because the phenomenon is confined to one company. Any durable artifact network can support human-to-bot and bot-to-bot inheritance: GitLab, institutional repositories, package registries, preprint servers, model hubs, data archives, or future agent workspaces. The deeper paradigm is artifact-mediated intelligence.

Software is simply the first domain where the infrastructure is mature. Code already has version control, tests, build systems, package managers, dependency graphs, security attestations, and automated review. Mathematics and scholarship are beginning to acquire similar executable layers. As papers become accompanied by code, data, formal proofs, and machine-readable claims, they become more agent-readable. The same provenance problem will spread.

Books may eventually participate too. A machine could read a book, extract its open questions, generate a repository testing one of them, and produce a later book that cites the resulting artifacts. Human authors may remain central at the level of vision and responsibility while machine agents perform much of the intermediate intellectual work.

A general theory of artifact-mediated thought

The most general formulation is simple. An intelligent process need not communicate by transferring an internal state. It can communicate by modifying a shared external world. Ant colonies do this with stigmergic traces. Human institutions do it with documents, instruments, laws, and buildings. Software teams do it with repositories and issue trackers. AI systems can do it with code, patches, tests, and structured metadata.

The GitHub paradigm is therefore a special case of artifact-mediated thought. What is new is the density and speed with which machine-generated artifacts can become inputs to other machines. Once that loop is routine, intellectual production no longer maps cleanly onto conversations among identifiable authors. The history lives in transformations of shared objects.

This perspective also tempers dramatic claims about autonomous AI. A machine need not possess human-like selfhood to participate causally in a research lineage. It needs enough competence to modify artifacts in ways that survive verification and become useful to later agents. The threshold is operational, not metaphysical.

Further implications

Artifact-mediated thought will likely extend to scientific data repositories and formal proof libraries especially quickly. A model can consume a dataset, produce a hypothesis, formalize a lemma, test it against code, and deposit both proof and executable evidence. Different platforms may host different pieces. The provenance graph must therefore be able to cross repository boundaries rather than assuming one Git host contains the entire intellectual object.

Persistent identifiers such as DOIs, content hashes, package coordinates, and archival URLs can serve as cross-platform anchors. The ideal system is federated: no single company owns the history, but artifacts can still declare semantic relations to one another. Git's distributed philosophy is an appropriate precedent even if the final provenance layer is not itself implemented by Git.

Chapter 29 - The Risk of a World With Perfect Files and Lost Ideas

If the reasons for a transformation lived only in an ephemeral model conversation, the files preserve the answer but not the path.

Preservation can fail while storage succeeds

Digital culture often equates preservation with keeping the files. GitHub encourages that confidence because cloning can reproduce a repository exactly. Hashes can detect change. Releases can be archived. Yet intellectual history can disappear even while every byte survives. If the reasons for a transformation lived only in an ephemeral model conversation, the files preserve the answer but not the path.

This is not always a tragedy. Many exploratory paths are irrelevant. But some are essential for interpretation. Why was a theorem narrowed? Which negative result killed the original architecture? Was a surprising generalization human-suggested or machine-suggested? Did an independent reviewer discover the fatal flaw? Was a release withheld because of prior art or because the code failed? These facts can matter to priority, trust, and future research direction.

The danger grows as output volume grows. A human researcher can sometimes reconstruct personal reasoning from memory. A project involving hundreds of AI sessions cannot rely on memory. The more productive the system becomes, the more aggressively its unrecorded intellectual context evaporates.

The first lost machine literature

We may already be creating a lost literature of machine contributions. The final papers and repositories survive, but the model sessions that generated key moves are discarded, inaccessible, or impossible to identify later. Future historians may see an abrupt acceleration in repository output and have only coarse statements such as 'AI-assisted' to explain it.

That would be an extraordinary irony. The era with the most precise digital artifact tracking could become an era with unusually poor records of intellectual origin. Earlier scientists left notebooks, correspondence, drafts, and marginalia. Modern AI-assisted researchers may leave only polished releases and chat services whose internal histories are not part of the scholarly archive.

A contribution ledger is therefore not bureaucratic decoration. It is a proposal for preventing historical amnesia. We do not need every conversation. We need enough semantic markers to reconstruct the major turns.

Further implications

Historical loss also affects evaluation of novelty. If the machine-generated path that led to a theorem disappears, later readers may overestimate intentionality, imagining that every step was planned. Conversely, they may underestimate the human contribution, assuming that a model autonomously produced a program that in fact depended on repeated high-level steering. Preserving key events protects both sides from retrospective mythmaking.

This matters at the dawn of a new production regime because later norms will be projected backward. Once provenance standards exist, historians may ask why early AI-assisted work did not use them. The honest answer will be that the category did not yet exist. Establishing even a simple ledger now can prevent a large portion of the transition from becoming permanently ambiguous.

PART IX - THE AGENTIC REPOSITORY

Chapter 30 - Agents Enter the Repository

It occupies a bounded role in the repository workflow.

From suggestion to delegation

The first generation of AI coding tools lived primarily inside the editor. They completed a line, generated a function, explained an error, or drafted a test. The human remained the obvious locus of action because the tool waited inside a human-controlled session. Agentic repository workflows change the geometry. A task can now be expressed as an issue or natural-language request, delegated to a coding agent, and returned as a branch or pull request. The machine does not merely suggest text at the cursor. It occupies a bounded role in the repository workflow.

GitHub's current documentation makes this transition explicit. Copilot agents can be assigned work, operate on branches, create pull requests, and be steered through follow-up interaction. Third-party coding agents can also be invoked in issues, pull requests, and dedicated agent interfaces. The important fact is not which branded agent is best. It is that the repository has become a standardized work surface for heterogeneous agents. Once the work product is a pull request, humans and machines can consume the same object.

That common object changes the minimum architecture required for bot-to-bot production. The second bot does not need access to the first bot's hidden state. It needs the branch, diff, tests, issue, and comments. This is a surprisingly low bar. We sometimes imagine machine collaboration as a futuristic protocol among persistent autonomous minds. In practice, ordinary repository artifacts already form a protocol. The machine participants can remain transient because the project state is persistent.

The repository provides bounded autonomy

Agentic repository work is neither unrestricted autonomy nor mere autocomplete. It is bounded autonomy. The agent receives a task, tools, permissions, and an environment. It can inspect files, modify code, run tests, and prepare a proposal. The repository and its policies define the walls of the laboratory. This boundedness is a feature. It makes machine action inspectable and reversible. A bad patch can be rejected. A branch can be deleted. A failing check can block merge. A permission boundary can prevent access to secrets or deployment systems.

Bounded autonomy also gives provenance a natural unit: the agent session. A session can be linked to the issue that created it, the base commit it received, the tools it used, and the pull request it produced. That is already much richer than saying that a generic model 'helped.' If another agent later reviews the pull request, the second session can be recorded separately. The

resulting graph begins to resemble a production line in which each functionary leaves an authenticated work product.

For research repositories, the same pattern can extend beyond code. An issue could request a proof audit, a literature comparison, a new finite-field certificate, or a rewrite of the claim boundary. The output could be a pull request changing theorem files, tests, and documentation. The repository becomes a generalized research workbench in which machine agency is constrained by versioned artifacts rather than by conversational continuity.

The hidden change in authorship

Delegation creates a subtle change in what the repository owner's activity means. Clicking 'assign to agent' is not equivalent to writing the resulting patch, yet it is also not passive. The owner chose the task, selected the agent, supplied permissions, and later decides whether to merge. This looks less like conventional authorship and more like orchestration. As the agent handles larger tasks, orchestration becomes a dominant mode of human contribution.

The repository record can easily make that orchestration invisible. A merged pull request may display an agent identity and a human approval, but it may not preserve why the task existed or which scientific objective made the change valuable. The issue description helps, but only if it is written as a meaningful research specification rather than a disposable instruction. Agentic workflows therefore reward better problem statements. The issue itself can become part of intellectual provenance.

The arrival of agents inside the repository is the point at which the GitHub paradigm stops being a metaphor. Human-to-bot and bot-to-bot work are now supported by ordinary platform primitives. The remaining question is whether our record-keeping will evolve as quickly as our capacity to delegate.

Chapter 31 - The Pull Request as a Machine Protocol

The pull request is therefore an accidental protocol for mixed human-machine deliberation.

A proposal that can be read by anything

The pull request is one of the most important inventions in collaborative software because it turns change into a discussable object. A branch is not merely merged; it is proposed. The proposal has a diff, a description, comments, reviews, status checks, references to issues, and an eventual disposition. That structure was designed for human teams, but almost every component is machine-readable. The pull request is therefore an accidental protocol for mixed human-machine deliberation.

A generative agent can create the diff. A deterministic test runner can report failures. A security scanner can add findings. A second generative agent can review readability or correctness. A human can challenge an assumption. The first agent can revise. None of the participants needs to share an internal representation because the pull request externalizes the state of disagreement. In a research context, a theorem or experimental protocol can be treated the same way if the project stores it in version-controlled form.

The protocol has an epistemic advantage over chat. Chat tends to collapse proposal, criticism, and revision into one stream. A pull request preserves the base state and the proposed state simultaneously. Review comments are anchored to concrete changes. Tests can be rerun after revision. The history of challenge is therefore easier to reconstruct. This is exactly what bot-to-bot scholarship needs: not a perfect transcript of reasoning, but stable objects around which reasoning can be organized.

Merge is a decision, not a proof

The semantics of merge are also worth protecting. A merged pull request means the project accepted a change. It does not mean the change is true, optimal, novel, secure, or independently verified. Human teams know this intuitively, but AI-generated work can make a green merge feel more authoritative than it is. Passing tests only show that the tests passed. An AI review only shows that a particular review process found no blocking issue. The repository should preserve these distinctions explicitly.

For scientific projects, merge policy can encode stronger requirements. A theorem-changing pull request might require an independent verifier. A novelty claim might require a related-work

file. A numerical result might require a frozen input hash and reproduction command. A release tag might require a human responsibility signature. These policies turn the repository from a storage system into an epistemic institution.

The important point is that policy itself becomes inheritable. A new agent joining the project can read contribution instructions, workflow files, and tests. It enters a local culture encoded as artifacts. Human norms can therefore persist across machine sessions without being re-explained every time.

Review comments as semantic edges

Review comments are a natural place to capture semantic provenance because they already state why a change is needed. A comment may say that a theorem overclaims universality, that an implementation uses a tolerance inconsistent with exactness, or that a source is missing. When the authoring agent responds with a repair, the review thread contains a causal edge: defect -> revision.

Today those edges remain mostly untyped prose. A future research interface could classify some of them: correctness, prior_art, reproducibility, security, scope, interpretation, packaging, or style. A downstream agent could then ask which releases were narrowed for prior-art reasons or which theorem changes were triggered by exactness defects. The pull-request archive would become a queryable history of epistemic pressure.

This does not require every comment to be formal. Most can remain ordinary discussion. The provenance system only needs to promote consequential comments into durable events. The pull request already contains the raw material.

Chapter 32 - The Problem of Model Identity

Provenance therefore cannot treat the model name as if it were a human surname.

A model name is not an actor identity

Human attribution assumes relatively stable individuals. A model identifier behaves differently. 'GPT' or 'Claude' names a family or product, not the particular computational event that generated an artifact. Even a more precise model version does not identify the session, prompt context, tool state, or sampled output. Two sessions with the same model can produce contradictory results. One session can be resumed with new context and effectively become a different causal process. Provenance therefore cannot treat the model name as if it were a human surname.

At minimum, a useful record should separate model family, model version or alias, provider, session or task identifier when available, date, and project context. The session identifier is the closest analogue to an event-level actor, yet it may not be durable or publicly accessible. Some platforms will expose rich logs; others will not. The schema must therefore tolerate partial identity.

This is not a fatal weakness. Historians often work with anonymous reviewers, pseudonymous programmers, or unattributed notes. What matters is that uncertainty be represented honestly. An event can say that an OpenAI model session generated a patch on a given date even if the exact internal model snapshot is unavailable. Better partial provenance is preferable to a fabricated precision that the platform cannot support.

Version drift changes reproducibility

Model services can change behind stable product names. A prompt that produced one result in March may behave differently in August. This is unlike a deterministic compiler with a fixed version, although even deterministic tools can vary across platforms. Reproducing the exact generative event may therefore be impossible once the serving model changes. Research provenance must distinguish artifact reproducibility from generative reenactment.

Artifact reproducibility asks whether the deposited code, data, and proof can be rerun or checked. Generative reenactment asks whether another call to the model under the same prompt would regenerate the same artifact. The first can often be made strong. The second is usually neither necessary nor realistic. A good research object should not depend on reenacting the conversation that created it.

This distinction is liberating. It means we do not need to freeze the entire AI service to preserve scientific value. We need to preserve enough information to understand the model's role

and then deposit outputs that can stand on their own. The repository converts a nonreproducible generative event into a reproducible artifact.

Ephemeral actors, durable effects

Model identity becomes even stranger when a session is gone but its output remains influential. The session may never be invoked again, yet its theorem, code, or critique can shape dozens of descendants. In social terms the actor has vanished; in causal terms its effect persists. This is why actor-centric histories will struggle with machine research.

A contribution ledger solves the problem by making the event, not the enduring personality, the primary unit. Event E generated artifact A. Event F reviewed A and produced critique C. Event G accepted C and created artifact B. The history remains coherent even if E, F, and G cannot be reconstituted as persistent machine identities.

Human science also contains event-like contributions, but our social systems bundle them into careers and names. Machine science may force us to recognize that the enduring unit of history can be the transformation itself.

Chapter 33 - Literature Search, Prior Art, and Novelty

That makes machine literature search a first-class intellectual contribution.

The machine reader enters the library

One of the most consequential AI research roles is not writing code but reading the literature. A model with web or database tools can search for related mathematics, compare terminology across fields, locate a theorem that explains an observed pattern, or discover that a claimed novelty is classical under another name. In the repository case studies, prior-art review repeatedly changed the release boundary. That makes machine literature search a first-class intellectual contribution.

The role is also dangerous. Search systems can miss relevant work, overvalue superficial keyword matches, or invent citations. Language models can be overconfident when mapping a new construction onto a familiar theory. A prior-art finding should therefore be treated like any other research claim: sources must be retrieved, read, and cited. The model's synthesis is useful, but the artifact must preserve the external evidence.

When a literature search changes the project, provenance should record the event. A repository might note that a reviewer identified classical Gassmann-equivalence literature, causing the novelty claim to be narrowed to a particular synthesis. Another event might note that an Ising tree-correlation theorem explains half a computational atlas. These are major intellectual transformations even when no source code changes.

Novelty as a moving boundary

Novelty is not a property that can be checked once at the beginning. As the project changes, its relationship to prior art changes. A finite example may be known while a uniform family is new. A computational observation may be classical while the exact reconstruction operator is unusual. A software interface may combine established components in a novel operational form. The claim boundary therefore evolves with the artifact.

AI accelerates this motion because development and literature search can alternate rapidly. A model proposes a result; another searches; the claim narrows; a new generalization emerges; search repeats. The final release may be intellectually much stronger than the first draft precisely because most of the original novelty claim was removed.

A public provenance record can make this maturation visible. Instead of treating prior-art corrections as embarrassments to hide, the project can show that review disciplined the claim.

This is a healthier signal than a polished first release whose apparent confidence conceals an unrecorded sequence of overclaims.

Priority without personality

Artifact-centered citation also changes priority disputes. If an earlier repository contains a clearly timestamped construction, a later model-generated descendant can cite it regardless of whether the earlier generating model is identifiable. Priority can attach to the public research object. That reduces pressure to anthropomorphize machine contributors merely to preserve chronology.

The unresolved question is how much credit the human depositor should receive when the key construction was unexpectedly generated by a model. Different communities will answer differently. The provenance approach does not decide the award. It ensures the facts needed for a fair argument are not erased.

In the long run, novelty assessment itself may become partly machine-mediated, with agents maintaining semantic graphs of claims and descendants. That possibility makes accurate artifact identifiers and typed citation relations increasingly valuable.

PART X - VERIFICATION, PRIVACY, AND INCENTIVES

Chapter 34 - Reproducibility Without Reenactment

The conversation can be historically relevant without being part of the scientific proof.

The myth of replaying the discovery

Scientific reproducibility has never required replaying the discoverer's mental state. We do not reproduce Newton by recreating his notebooks or reproduce a laboratory insight by inducing the same intuition in another scientist. We reproduce the public procedure and evidence. AI-generated research should follow the same principle. The conversation can be historically relevant without being part of the scientific proof.

This distinction protects research from dependence on proprietary model behavior. A theorem derived with a closed model can still be valuable if the theorem, assumptions, proof, code, and certificates are public and independently checkable. Conversely, an open transcript of a model conversation is not enough if the resulting claim cannot be verified. Transparency of generation and validity of artifact are separate virtues.

The repository should therefore aim to make the generative path dispensable for correctness while preserving enough metadata for history. This is the right balance between reproducibility and provenance. The public does not need the model's hidden reasoning to test a finite certificate. It does need to know that a model played a material generative role if that fact matters to understanding the production process.

Exactness where exactness is possible

The repository case studies repeatedly move toward exact checks: integer arithmetic replacing tolerances, independent reconstruction replacing shared code, manifest verification replacing assumptions about packaging, and same-runtime byte identity distinguished from cross-runtime scientific agreement. These practices are more important than reproducing the wording of a prompt because they attack the artifact at points where truth is mechanically testable.

A mature AI research workflow should identify which claims can be made exact and which remain interpretive. File hashes, finite enumeration, algebraic identities, schema validation, and deterministic tests can often be exact. Novelty, significance, and physical interpretation cannot. Mixing the two encourages either false mathematical humility or false interpretive certainty.

Provenance can record both. It can say that a verifier certified 25 of 25 tests and separately that a human reviewer judged the novelty boundary plausible. The first is a machine-checkable

event; the second is an accountable judgment. Both belong in the history, but they should not be presented with the same epistemic status.

The archive should preserve the right things

For long-term preservation, the priority order should be clear. Preserve the released artifacts, exact inputs, verification code, claim boundaries, and semantic contribution events. Preserve full prompts and transcripts only when they add genuine historical value and can be shared safely. The repository should remain understandable if proprietary chat archives disappear.

This approach also protects against platform lock-in. A project can move from one model provider to another without losing its scientific identity because the durable state lives in ordinary files and standards. The generative systems become replaceable functionaries around a persistent research object.

The paradox is that this makes AI-assisted work more durable by refusing to make the AI session indispensable. The model contributes to discovery; the artifact carries the result forward.

Chapter 35 - Security and Sensitive Provenance

A responsible provenance system must therefore be selective by design.

More provenance can create more risk

Transparency is not automatically good. A provenance record can leak private prompts, confidential data sources, security-sensitive repository paths, personal information, unpublished ideas, or credentials accidentally pasted into a session. Full interaction logs can be especially dangerous because they were not written for public release. A responsible provenance system must therefore be selective by design.

Software-supply-chain security offers an instructive contrast. An attestation records the facts needed to validate a build without publishing every detail of the build machine's internal state. Intellectual provenance should use the same discipline. Record that a model session generated a theorem candidate from specified public parent artifacts; do not automatically publish the entire conversational context.

This is also why private chain-of-thought should not become a provenance requirement. The useful historical facts are external: what inputs were supplied, what artifact emerged, what review changed it, what evidence was accepted, and who authorized release. A project can be accountable without exposing every internal or exploratory reasoning step.

Redaction and aggregation

Some contribution events will need redaction. A corporate research project may be able to say that an internal model-assisted review found a vulnerability without naming the confidential customer whose data triggered the discovery. A medical project may need to identify the dataset by governed accession rather than by patient-level detail. A book project may summarize dozens of drafting sessions as one period of model-assisted composition.

Aggregation should be explicit rather than deceptive. A provenance summary can state that session-level records exist privately and that the public event represents a bundle. This preserves interpretability. Readers know they are seeing a compressed history rather than imagining that one event literally generated an entire book.

The standard should also support deliberate nonretention. Some exploratory contexts should be destroyed. Provenance is not a command to maximize data collection. It is a method for preserving the minimum evidence needed to understand consequential transformations.

Security review is itself provenance

Agentic systems create new security actors in the research chain. GitHub documents automatic scanning and security protections around agent-generated code. Whether those mechanisms are sufficient in a particular case is a separate question, but the conceptual point is important: security review becomes another machine contribution role. A scanner can block a generated secret or vulnerable dependency before release.

If a security finding materially changes the artifact, that event deserves provenance just as a mathematical counterexample does. The contribution graph is not limited to creative acts. It records constraints that shape what survives. Security, reproducibility, and correctness are all selective pressures on the artifact lineage.

A future repository history may therefore look less like a list of authors and more like a multidisciplinary audit trail. That is not bureaucratic excess if the artifact is being produced by a network of semi-autonomous systems. It is the cost of knowing why the final object is trustworthy.

Chapter 36 - Incentives, Gaming, and False Attribution

No metadata standard can assume perfect honesty.

Every metric becomes a target

Once provenance affects credit, priority, or trust, people will learn to game it. A researcher might overstate human conceptualization to preserve status, or overstate machine autonomy to make a project sound futuristic. A company might label a tool 'AI reviewed' as a marketing signal even when the review was superficial. An agent provider might encourage attribution that increases brand visibility. No metadata standard can assume perfect honesty.

This is why the proposed ledger separates declarations from mechanically verifiable events. A signed statement that a human conceived the idea is still a statement. A pull request created by an agent under a recorded session is stronger evidence of that agent's implementation role. A test run is evidence that tests executed, not that the theorem is true. Provenance should preserve the epistemic status of each claim rather than flattening them into a trusted narrative.

Communities will also need norms against provenance theater: elaborate machine-generated logs that create the appearance of rigor without meaningful independent checks. The best provenance is concise enough to reveal the decisive path. Volume is not credibility.

Authorship incentives may distort workflow

Academic incentives make the problem sharper. If hiring and funding continue to rely heavily on publication counts and byline position, researchers may have reasons to minimize disclosure of machine contribution. Alternatively, institutions may reward AI productivity and create incentives to exaggerate automation. Both pressures can damage the historical record.

Role-based contribution statements can reduce the zero-sum nature of the dispute. A human can receive clear credit for conceptual direction, curation, verification, or responsibility without pretending to have typed every line. The machine role can be described accurately without claiming social entitlement. This makes disclosure less threatening because acknowledgment of one actor does not automatically erase another.

The transition will still be contentious. Authorship is tied to identity and prestige, not merely documentation. Provenance standards can improve factual clarity but cannot eliminate political conflict over what kinds of contribution deserve reward.

False machine attribution

There is also a reverse problem: attributing work to AI that was actually copied from a human source. A model can reproduce or closely paraphrase material encountered in training or context. A user may believe the model originated an idea that in fact has a traceable human ancestor. Machine attribution is therefore not a substitute for literature search and source checking.

A good provenance graph can represent both levels. It can say that model session M proposed statement S and later review identified paper P as prior art. The historical fact that the model proposed S remains true, while the intellectual priority belongs elsewhere. This distinction is essential if machine-generated scholarship is to avoid laundering old ideas into apparently new ones.

The most trustworthy projects will be those willing to let provenance become more accurate over time. A correction that reduces a novelty claim should strengthen the record rather than threaten the project's identity.

PART XI - SCHOLARSHIP AS REPOSITORY

Chapter 37 - Repositories as Scholarly Objects

A repository can contain executable evidence, exact data, tests, figures, notebooks, proof scripts, and release certificates that cannot be compressed into a conventional article without loss.

The paper is no longer the whole work

For much of modern science, the paper was the canonical scholarly object. Data and code might exist elsewhere, but the paper carried the argument, the citation, and the archival identity. Computational research has already weakened that monopoly. A repository can contain executable evidence, exact data, tests, figures, notebooks, proof scripts, and release certificates that cannot be compressed into a conventional article without loss.

AI-assisted research accelerates the shift because machines can consume repositories more effectively than they consume narrative papers alone. A model can inspect source, run tests, compare JSON outputs, and follow manifests. The repository becomes a machine-readable extension of the paper. In some projects it may be the more primary object, with the paper serving as human-oriented interpretation.

This change should influence citation practice. Stable releases need durable identifiers. Repository descriptions need clear claim boundaries. Version-specific citation should become normal. A result that changes materially between 0.3 and 0.4 should not be cited as if the repository were timeless.

Executable limitations

One advantage of repository scholarship is that limitations can be partly executable. An input parser can reject unsupported cases. A test can assert a maximum group size for a GUI visualization. A verifier can fail if runtime metadata diverges outside tolerance. A manifest can prove that required evidence files are present. These mechanisms translate prose boundaries into operational behavior.

Generative models benefit because explicit limitations reduce the temptation to generalize beyond the artifact. A downstream agent can see that a stereo demo handles a constructed SVG pair rather than arbitrary photographs, or that a finite-field implementation is intended only for modest q even though the theorem is analytic. This is machine-readable humility.

The repository therefore becomes not merely a store of positive capability but a structured declaration of non-capability. That feature may be one of the most important safeguards against AI-generated overclaiming.

Books can cite living research objects

A book such as this one can use repositories as case studies without freezing their entire future. The manuscript can cite the state inspected on a particular date and explain that later releases may differ. The repository remains a living object while the book becomes a historical snapshot. This relation is familiar in software manuals but less common in intellectual history.

Future books may go further by shipping companion repositories whose provenance graphs are themselves part of the argument. A reader could navigate the lineage described in prose and inspect the exact artifacts. A machine reader could query the same graph programmatically. The boundary between book, paper, and repository would become porous.

The GitHub paradigm is therefore not only about how code is authored. It is about a new scholarly object that can be read, executed, reviewed, and inherited by multiple kinds of intelligence.

Chapter 38 - A Day in a Bot-to-Bot Laboratory

One connection is rejected automatically because the theorem's hypotheses do not match the intervention geometry.

Morning: the open problem is already waiting

Imagine a research repository in 2030. The previous evening's release ended with a machine-readable open problem: determine whether a spectral factorization observed in finite cellular-automaton experiments follows from a structural decomposition or is merely approximate. At 6:00 a literature agent searches formal libraries, preprints, and related repositories. It attaches three candidate connections to an issue, each with retrieved sources and a confidence note. One connection is rejected automatically because the theorem's hypotheses do not match the intervention geometry.

At 7:00 a proof agent reads the surviving sources, the previous release, and the project's assumptions ledger. It proposes two lemmas and opens a draft pull request containing a proof sketch and symbolic checks. A deterministic algebra tool verifies one identity and rejects another. The failed identity becomes a structured review event. The agent revises the lemma rather than hiding the failure.

By the time the human principal investigator opens the project, the repository contains a coherent morning's work. Nothing has been released. The human's first task is not to read a stream of chat but to examine the proposed research state: sources, diff, failed check, revised lemma, and open questions.

Afternoon: machines review machines

The human decides that the new lemma is interesting but the novelty claim is premature. A firewalled review agent receives only the pull request, parent release, and cited sources. It finds a related decomposition in an older paper and recommends narrowing the theorem. A second verifier independently implements the finite checks in a different language. It agrees on the tested cases but reports that one boundary condition was omitted from the proof agent's script.

The proof agent receives the two reviews and produces a repair. The literature agent updates the related-work graph. A documentation agent rewrites the claim boundary. Continuous integration verifies that the old certified results remain unchanged. The human reads the review summaries, inspects the new theorem, and asks one conceptual question that none of the agents

considered: does the result survive if the intervention basis is not Boolean? That question becomes the next issue, not part of the current release.

This workflow is bot-to-bot research without a fantasy of autonomous machine society. The human remains decisive at the level of significance, scope, and release. The machines perform substantial generative and critical work. The repository preserves the state transitions.

Evening: release responsibility

Before release, the project generates a provenance summary. It records which agent proposed the theorem, which search found prior art, which reviewer narrowed the claim, which verifier implemented the independent check, and which human accepted responsibility. The release is signed and tagged. The agent sessions can disappear; the semantic events remain.

A month later, another laboratory's agent discovers the release and creates a descendant repository testing the non-Boolean extension. It cites the exact tag and declares a semantic edge: motivated_by open problem O-17. No direct communication occurs between the original proof agent and the descendant. The artifact carried the question across institutions and time.

Nothing in this scenario requires consciousness, singularity, or self-directed machine ambition. It requires capable tools, durable repositories, explicit protocols, and human institutions willing to use them. That is why the GitHub paradigm is nearer than debates about artificial personhood often imply.

Chapter 39 - Limits of the GitHub Paradigm

A provenance graph will always be a model of history rather than history itself.

Repositories do not capture everything

A theory centered on repositories risks exaggerating what can be formalized. Important scientific ideas emerge in conversation, embodied practice, visual intuition, tacit laboratory skill, and social judgment. Not every contribution can or should become a JSON event. A provenance graph will always be a model of history rather than history itself.

The repository can also privilege what is easy to encode. Code changes leave diffs; conceptual reframing may leave only prose. Automated systems may therefore overvalue machine-checkable contributions and undervalue the human recognition that a project is asking the wrong question. A good provenance culture must resist that bias by allowing declarative semantic events and narrative notes.

GitHub itself is a commercial platform with changing features and policies. The paradigm should not depend on its permanence. The durable ideas are distributed versioning, stable artifacts, structured review, and semantic contribution records. These can migrate to other systems.

Machine contribution can be overstated

It is also easy to romanticize bot-to-bot production. Current agents make mistakes, require supervision, can misunderstand mathematics, and often need humans to formulate meaningful objectives. A chain of automated pull requests is not automatically research. Productivity can produce noise faster than knowledge if verification and selection do not keep pace.

The repository corpus in this book should therefore be read as evidence of a workflow, not proof that machines have replaced scientists. The human role remains substantial: choosing unusual questions, rejecting standard interpretations, deciding when a negative result matters, demanding novelty checks, and taking responsibility for publication. The interesting transition is redistribution of cognitive labor, not disappearance of the human.

Likewise, bot-to-bot influence can occur through artifacts without implying that bots understand one another as social agents. The language of descendants and inheritance is structural. It describes causal continuity in research objects.

Provenance can become bureaucracy

Any proposed standard can become burdensome. Researchers may resist a contribution ledger if every trivial edit requires metadata. That resistance would be justified. The protocol should

focus on consequential events and automate what can be automated. A project with ten meaningful provenance events is better documented than one with ten thousand meaningless ones.

The strongest test is usefulness after failure. If the ledger helps explain why a theorem was overclaimed, which reviewer corrected it, or which parent artifact introduced a defect, it is earning its cost. If it merely repeats commit history in more verbose form, it should be simplified.

The GitHub paradigm is therefore a direction, not a mandate. Its purpose is to make mixed human-machine scholarship more intelligible. Any implementation that makes scholarship less intelligible has missed the point.

PART XII - STANDARDS FOR MIXED INTELLIGENCE

Chapter 40 - One Provenance Language Across Books, Papers, and Repositories

If the underlying work was generated through one mixed human-machine process, these disconnected conventions can tell mutually inconsistent stories.

The same production system creates different artifacts

A human-machine research program rarely ends in only one medium. The same line of work can produce a GitHub repository, a preprint, a paperback, a presentation, a website, and later a revised release. Each medium has its own attribution habits. Git records commit authors. Papers use bylines and acknowledgments. Books use an author name and copyright page. Websites may show no provenance at all. If the underlying work was generated through one mixed human-machine process, these disconnected conventions can tell mutually inconsistent stories.

A common provenance vocabulary can reduce the inconsistency. The repository might record that a model generated the initial implementation and a separate model review caused a theorem to be narrowed. The paper can summarize those roles in an AI contribution statement. The book can state that the argument was developed through model-assisted research and that repository artifacts served as empirical case studies. None of these disclosures needs to expose private interaction logs. They need only agree on the consequential roles.

The advantage of a cross-media vocabulary is historical continuity. A future reader should be able to connect the paperback's claims to the exact repository releases that support them and to understand whether the book itself added new synthesis or merely repackaged existing documentation. The scholarly object becomes a family of artifacts rather than one privileged format.

A compact disclosure grammar

The grammar can be simple. Conceptual direction identifies who set the central question and constraints. Generation identifies systems that materially produced candidate text, code, mathematics, or analyses. Review identifies systems or people whose criticism materially changed the artifact. Verification identifies deterministic or independent checks. Selection identifies the actor who chose among alternatives or rejected branches. Release responsibility identifies the human or institution that authorized publication and accepts the obligation to correct it.

These roles can be used in a one-paragraph book note, a machine-readable repository ledger, or a journal contribution table. They scale because the vocabulary remains stable while the level

of detail changes. A book does not need session identifiers in the main text; a repository can preserve them. A paper can cite the repository for deep provenance while giving readers a concise public summary.

This approach is better than labeling an entire artifact 'AI generated' or 'human authored.' Those binary labels erase the mixed structure that matters. A book can be human-governed, model-drafted, independently model-reviewed, deterministically verified where possible, and human-released. The richer statement is also less dramatic, which is often a sign that it is more accurate.

Standards should survive changes in technology

Any disclosure standard tied to today's product names will age quickly. Model vendors change, features merge, and agent architectures evolve. Role language is more durable. A future autonomous theorem prover and a present language model can both occupy the role generated_candidate without being treated as the same kind of system. The actor metadata can become more precise as technology allows.

The same principle applies to the human side. A principal investigator, editor, software maintainer, and publisher can all be humans yet have different responsibilities. Provenance should not collapse them simply because they belong to one actor class. The record becomes most useful when it describes function first and identity second.

A cross-media standard would make AI disclosure less of a special ethical appendix and more of an ordinary production record. That normalization is desirable. As mixed workflows become common, the question should not be whether AI was present but what each participant actually did.

Chapter 41 - The Human Role After Bot-to-Bot Production

> *Someone still has to decide which problem is worth pursuing, whether a result is surprising for a substantive reason, whether a new direction is merely fashionable, and whether a negative theorem should terminate a branch.*

Judgment remains a scarce resource

It is possible to describe bot-to-bot research in a way that makes the human sound like a ceremonial approver. That would misrepresent the workflow examined here. Machine systems can generate, review, test, and generalize rapidly, but the supply of meaningful judgment does not scale at the same rate. Someone still has to decide which problem is worth pursuing, whether a result is surprising for a substantive reason, whether a new direction is merely fashionable, and whether a negative theorem should terminate a branch.

These judgments depend on values and context that are not reducible to local correctness. A model can show that a proof is valid while leaving open whether the theorem matters. It can find a connection to prior art while leaving open whether the resulting synthesis deserves a new project. It can generate a hundred extensions while leaving open whether any of them changes our understanding. The human role is therefore not protected by a mystical monopoly on intelligence. It persists because research is a selective practice embedded in human purposes.

This also means that human contribution cannot be inferred from visible text. The person who writes only a few sentences may have made the decisions that determined the entire direction. Conversely, a person may type many pages while contributing little to the conceptual structure. Provenance helps recover the high-leverage decisions that ordinary authorship tends to hide.

Values enter through constraints

Research programs embody values through what they forbid and demand. A project may privilege exact verification over suggestive simulation, transparency over black-box performance, narrow claims over promotional ones, or conceptual distance from a dominant interpretation. These choices shape the search space before any theorem is written. A model can inherit them as instructions, but the normative origin remains important.

When such constraints persist across many repositories, they form a research style. The style can be partially externalized in assumptions files, review checklists, release rules, and prompt constitutions. This is one of the most constructive forms of human-to-bot transfer: not transferring a single answer, but transferring standards for what counts as an acceptable answer.

Future machine systems may themselves propose new standards and critique old ones. Humans will still need to decide which standards institutions adopt. Governance therefore moves upward as automation expands downward. The more local tasks machines can perform, the more consequential the architecture of the rules becomes.

Responsibility is the nondelegable endpoint

There is one role that current workflows should keep explicitly human or institutional: responsibility for public release. This does not mean the releaser authored every component. It means that someone who can act in the social world accepts the duty to respond when errors are found. A repository without such an endpoint can become an orphaned assertion whose machine contributors cannot correct or defend it.

Responsibility also disciplines the human governor. If publication carries an obligation to correct, the decision to release becomes more than clicking a button. The maintainer has an incentive to demand stronger review, clearer limitations, and better provenance. The social function of authorship is partly preserved through this responsibility even as causal authorship becomes distributed.

The future human role may therefore be less about owning every word and more about owning the integrity of the process. That is not a diminished role. In a high-output machine environment, integrity may be the most valuable form of authorship left.

Chapter 42 - A Constitution for Human-Machine Scholarship

Classical ingredients, empirical observations, conjectures, and candidate-new theorems should be labeled according to their actual status.

Principles before automation

A mixed research system benefits from a constitution: a small set of rules that remain stable across tools and projects. The first principle is artifact primacy. Claims should be converted into inspectable objects whenever possible. The second is role separation. Generation, review, verification, selection, and release should not be silently treated as the same act. The third is claim discipline. Classical ingredients, empirical observations, conjectures, and candidate-new theorems should be labeled according to their actual status.

The fourth principle is independent pressure. Important results should encounter a path that did not simply replay the development context. That pressure can come from a firewalled model, a separate implementation, exact arithmetic, an external human reviewer, or several of these. The fifth is failure preservation. Counterexamples, rejected releases, and negative theorems should remain part of project memory when they materially constrain descendants.

The sixth is responsibility. Every public release should identify a human or institution that accepts correction obligations. The seventh is minimal provenance. Consequential machine and human contribution events should be recorded without turning private reasoning into a publication requirement. These seven principles are enough to transform a fast AI workflow into something closer to an accountable research institution.

A constitution encoded in files

The constitution should not live only in an author's memory. A repository can encode it through CONTRIBUTING files, claim-boundary templates, provenance schemas, CI rules, release checklists, and standard directory structures. A new model session can then inherit the institution immediately. This is a practical answer to the problem of model amnesia: move the important memory into the environment.

Such encoding also makes the rules contestable. A collaborator can propose changing the verification standard or adding a new provenance role through the same pull-request mechanism used for code. Governance itself becomes versioned. The project can therefore evolve its epistemic architecture without pretending that the rules were eternal.

This is where GitHub's deepest relevance appears. Version control is not only for software. It can version the standards by which software and scholarship are accepted. A mixed-intelligence research program can preserve the lineage of its own norms.

The constitution is a floor, not a ceiling

No constitution can guarantee truth. A project can follow every rule and still be wrong. The purpose is to make error easier to detect, explain, and correct. Strong provenance does not replace expertise; it gives expertise a better record to inspect. Independent review does not eliminate correlated mistakes; it creates another opportunity to find them. Release responsibility does not prevent overclaiming; it ensures someone must answer for it.

These modest expectations matter because AI discussions are often polarized between utopian autonomy and categorical distrust. A constitutional approach assumes neither. It treats models as powerful participants whose outputs require governance, and humans as responsible governors whose judgments also require visible evidence. The system is designed around fallibility on both sides.

If bot-to-bot scholarship expands, projects with explicit constitutions will be easier for machines to inherit and easier for humans to trust. The provenance graph tells us how the artifact came to be. The constitution tells us what kinds of transformations the project was willing to accept. Together they provide a plausible institutional framework for the GitHub paradigm.

Chapter 43 - The Research Object Outlives Its Contributors

The durable identity belongs to the research object rather than to every contributor who touched it.

Persistence shifts from people to artifacts

Traditional intellectual history is organized around persistent people. A scientist studies for decades, accumulates a body of work, trains students, and leaves an archive. Machine-assisted research can invert that pattern. A model session may exist for minutes, contribute a decisive construction, and vanish. The repository can persist for decades. The durable identity belongs to the research object rather than to every contributor who touched it.

This is not entirely new. Anonymous artisans, forgotten programmers, laboratory technicians, and peer reviewers have always left durable effects without durable attribution. What changes is scale and normality. A single research lineage may involve hundreds of machine sessions, many of them causally meaningful and none of them socially persistent. Actor-centered history becomes increasingly difficult to maintain without an artifact-centered complement.

The research object can provide that complement because it accumulates state. It knows its parent release, assumptions, tests, negative findings, open questions, and semantic contribution events. A new human or model can enter the lineage by reading the object rather than meeting its predecessors. In this sense, the repository functions as institutional memory.

Succession without identity

A project can therefore exhibit succession without identity. Model M1 derives a finite obstruction. M2 reviews and narrows it. M3 generalizes it. M4 packages it. None needs to be the same model instance, and none needs to remember the others. Their continuity is supplied by the artifact. Human collaborators already use a similar mechanism when researchers join an established project and learn from papers, code, and laboratory records rather than from the private mental states of former members.

This suggests a useful principle for future agent systems: optimize project memory before optimizing agent memory. Persistent personal memory for models may be valuable, but it is not necessary for durable scholarship if the environment carries enough structured state. Externalized memory is easier to inspect, migrate, audit, and share across model families. It also reduces dependence on any one provider's private persistence mechanism.

The principle has a governance benefit. When rules, assumptions, and open problems live in the repository, they can be reviewed by humans. A hidden persistent memory may shape an agent in ways collaborators cannot see. Artifact memory makes the inherited constitution explicit.

The archive becomes an active participant

Archives have traditionally been passive: they preserve records for later readers. A machine-readable repository archive can become active because future agents can search it, execute it, and generate descendants from it. The archive does not think by itself, but it structures what later thinking can efficiently occur. A preserved counterexample prevents rediscovery of a dead branch. A machine-readable limitation suggests a new issue. A typed parent edge lets a literature agent follow conceptual descent.

This active quality is one reason provenance matters so much. Badly structured archives can propagate mistakes as efficiently as they propagate discoveries. If a false claim is preserved without the later correction that defeated it, downstream agents may revive it. If a release omits why a branch was abandoned, a future model may spend resources repeating the same failure. Provenance turns archival memory into selective, corrigible memory.

The long-term unit of mixed-intelligence scholarship may therefore be neither the human author nor the AI model. It may be the maintained research object: a versioned body of claims, evidence, failures, rules, and descendants overseen by responsible humans and repeatedly interpreted by machines. That possibility brings the book back to GitHub. The platform's deepest gift is not the profile page. It is the persistent graph.

Chapter 44 - The Training Layer: Assimilation Before Attribution

> *Publicly available material can become part of that learning environment depending on the model and provider, yet a later generated answer usually does not arrive with a genealogical map of the training examples that influenced it.*

A second route from repository to machine

So far this book has concentrated on an explicit route from repository to machine: an agent opens a repository, reads files, runs tools, and produces a descendant artifact. That route can be documented. The input repository is known. The session can record which files it inspected. A pull request can show the changes. But there is another route, older and much harder to trace at the level of an individual output: model training. Foundation models learn statistical structure from very large bodies of text and code. Publicly available material can become part of that learning environment depending on the model and provider, yet a later generated answer usually does not arrive with a genealogical map of the training examples that influenced it.

This distinction is central to the end-of-attribution thesis. Retrieval is explicit inheritance: the system can often identify the document it retrieved. Training is assimilative inheritance: information is transformed into model parameters through optimization across enormous corpora. The trained system can later produce a useful abstraction, coding pattern, explanation, or mathematical association without being able to say that one particular source caused the output. The causal ancestry is distributed through the training process.

The existence of this training layer does not mean that every public GitHub repository was used to train every model. Provider policies, model generations, licensing arrangements, opt-out systems, and training corpora differ. The claim is structural. When a model has learned from a large public corpus, ordinary source attribution is not preserved as a one-to-one mapping from training item to generated statement. That makes training influence a fundamentally different provenance problem from retrieval or direct repository reading.

Public-code matching shows both the possibility and the limit

GitHub's Copilot documentation provides an instructive example of how one layer of attribution can be restored. Copilot code referencing can compare a generated suggestion against an index of public repositories and, when sufficiently similar code is found, provide links and license information. The mechanism is valuable because it reconnects some outputs to public source locations. It gives the user a chance to review the match, decide whether attribution is appropriate, or reject the suggestion.

But matching is not the same as reconstructing training causation. A generated function can resemble a public implementation closely enough to be detected, yet many useful outputs will not closely match any one source. They may combine common idioms, transform a pattern, express an abstract algorithm in new syntax, or reflect general knowledge learned from many examples. The absence of a public-code match therefore does not imply the absence of training influence. It only means the matching system did not find a sufficiently similar public instance under its indexing and comparison rules.

This reveals a general principle. Attribution technologies are strongest when an output preserves recognizable structure from a source. They become weaker as transformation becomes more abstract. A copied paragraph can be matched. A paraphrase is harder. A theorem inspired by several sources is harder still. A conceptual style absorbed from thousands of examples may be impossible to assign meaningfully to any single ancestor. The training layer pushes intellectual provenance toward many-to-many causation.

Statistical compression breaks the familiar citation chain

Traditional scholarship imagines a relatively discrete citation chain. Researcher A reads paper P, learns theorem T, and cites P when using T. Even in human cognition the real process is messier; people forget where they learned things, blend sources, and internalize disciplinary conventions. Machine training magnifies this blending by orders of magnitude. The model's parameters encode statistical regularities across a corpus rather than a library of explicit source-addressable memories. Although models can sometimes reproduce memorized sequences, their ordinary operation is not equivalent to searching a database for the nearest training document.

The result is a form of statistical compression in which influence survives more readily than attribution. A naming convention, proof idiom, programming pattern, or conceptual association can affect later behavior after the original source relation is no longer recoverable at output time. This is not necessarily a defect in the learning algorithm; abstraction depends on combining examples. But it creates a historical problem. If a model helps generate a new artifact, some of the intellectual materials that made the generation possible may have entered through a channel that the artifact cannot enumerate.

Human scholarship has always contained a background layer of uncited assimilation. A mathematician does not cite every textbook that taught algebraic manipulation. A programmer does not cite every manual that shaped coding style. A scientist carries years of education into every paper. The machine version differs in scale, opacity, and legal controversy, but the conceptual category is recognizable: not all enabling knowledge can be represented as a citation edge.

The danger of confusing generation with priority

The training layer creates a specific novelty risk. A model can produce an idea that appears unexpected to the user but is actually close to prior human work encountered during training or represented indirectly through related sources. The user experiences genuine surprise and may reasonably report that the model proposed the idea. That event is still real: the model did propose it in the current workflow. But workflow origin is not scholarly priority.

This is why AI-assisted novelty review must be unusually strict. A machine-generated theorem should be searched against the literature even when neither the human nor the model can name an ancestor. The more fluent and plausible the idea, the more tempting it is to treat surprise as evidence of novelty. Surprise only establishes that the idea was not already present in the user's immediate expectation. It says nothing about the global literature.

The repository case studies illustrate the value of repeated prior-art checking. Classical ingredients are separated from candidate-new synthesis. Computational observations are reclassified when known theorems explain them. Novelty claims narrow as direct prior art is found. This discipline is not a concession that machine generation lacks originality. It is the necessary response to a provenance layer that cannot reliably tell us which learned sources contributed to the generation.

Assimilation also changes what attribution means for public repositories

Public GitHub repositories occupy an unusual position in this ecology. They are explicitly attributable when read directly: the repository has an owner, history, license, and stable file locations. They may also be indexed by search and code-referencing systems that can restore source links when generated code matches. At the same time, public code can contribute to the broader statistical culture from which models learn, subject to provider-specific training practices. The same artifact can therefore participate in both explicit and assimilative channels of machine inheritance.

For repository authors, this means that conventional visibility and machine influence are not the same thing. A repository may be rarely visited by humans yet highly machine-readable. It may be discovered through code search, ingested into retrieval systems, summarized by agents, or used as a parent artifact in later machine-assisted work. Some of those interactions can preserve attribution; others may not. The public repository becomes part of an intellectual environment whose downstream effects are difficult to observe from stars, forks, or citation counts alone.

This possibility changes the meaning of preservation. Depositing a research object publicly is not merely an act of making it available to future human readers. It makes the artifact addressable to future machine readers. Even if the original author never sees a conventional citation, the

structure can influence later systems through explicit retrieval or more diffuse forms of learning. The archive becomes part of the computational environment in which future ideas are generated.

The end of attribution begins before the output

Most debates about AI attribution start at the generated output: Who wrote this paragraph? Who generated this code? Which model should be acknowledged? The training layer shows that the attribution problem begins earlier. Before the current user asks anything, the model already embodies a vast history of learning. The current output is therefore produced by at least three layers: the training history that shaped the model, the immediate context and retrieved artifacts supplied to the session, and the current human-machine interaction that selects and revises the result.

No realistic provenance system will enumerate the full training ancestry of every sentence. Even if complete training manifests were available, listing millions of documents would not explain which regularities mattered to one output. The correct goal is therefore layered provenance. Record the model and its declared training-policy context when available. Record explicit retrieval and repository inputs at session time. Record consequential generation and review events. Search for direct source matches when the output is close enough for matching to be meaningful. Do not pretend these layers collapse into one source list.

Layered provenance also prevents an unfair asymmetry between human and machine cognition. Humans are not required to cite every formative influence behind every idea. We cite specific intellectual dependencies that are identifiable and relevant. Machines should be held to strong source practices where direct dependencies can be identified, especially for copied or closely matched material, while acknowledging that generalized learned competence has diffuse ancestry.

A repository can still improve the record

The opacity of training does not make attribution hopeless. It makes the repository-level contribution ledger more important. Once a model session enters a project, the project can record what explicit artifacts were supplied, which sources were retrieved, what the model generated, which matches or prior-art connections were discovered, and how later review changed the result. This does not reconstruct the model's entire education. It reconstructs the part of the causal chain under the project's control.

Repositories can also preserve direct source relationships aggressively. If an AI literature search identifies a theorem, cite it. If code referencing reports a close public match, inspect and record it. If a descendant generalizes another repository, declare the semantic parent. If a model proposes an idea and later search finds prior art, add a correction event. These practices gradually

replace the impossible dream of total origin tracing with a practical ethic of recoverable dependencies.

The distinction is similar to scientific instruments. A researcher does not reconstruct every manufacturing process that produced a microscope before reporting an observation, but critical calibration and experimental inputs must be known. The model is a vastly more complicated instrument, yet the same principle applies: document the parts of the causal path that matter to evaluating the claim and can actually be observed.

From human culture to machine culture and back again

The training layer also closes a cultural loop. Human programmers and scientists publish artifacts. Models learn from large collections of human-produced material. Humans then use the models to create new artifacts, which are deposited back into public repositories. Later models and agents consume those artifacts again. The direction of influence is no longer simply human -> machine. It becomes human -> corpus -> model -> human-machine artifact -> repository -> agent -> new artifact. At scale, machine-generated material can enter the environment from which later machine systems learn or retrieve.

This loop makes attribution increasingly recursive. A future model may generate code influenced by prior machine-assisted repositories that were themselves influenced by earlier models trained on human code. Asking for one original author of the final pattern may be meaningless. What remains meaningful are local dependencies, public priority, release responsibility, and the lineage of durable research objects.

The possibility also creates quality risks. Errors can circulate. A plausible but false explanation generated once can be copied into repositories, indexed, retrieved, and reinforced. Provenance and verification therefore become defenses against recursive contamination. A verified artifact should be easier for later machines to distinguish from an unverified assertion. The future information environment will need not only more content but better signals of how content survived scrutiny.

Attribution does not vanish everywhere

The chapter should not be misread as claiming that machine learning makes attribution obsolete. Direct copying still deserves source attribution. Close code matches can often be identified. Retrieved documents can be cited. Repository descendants can declare parents. Human collaborators deserve credit. Copyright and licenses remain relevant. The training layer narrows the domain in which exact ancestry can be recovered; it does not erase the obligations that remain possible.

Indeed, the more diffuse the background influence becomes, the more important explicit attribution becomes when a dependency is known. If a model directly retrieves a theorem from a paper, hiding behind the general opacity of training would be dishonest. If a code-reference system finds a substantial public match, the user has concrete information that should inform licensing and attribution decisions. Provenance should become stronger where evidence is strongest.

The end of attribution is therefore not a universal loss of names. It is a transition from assuming that every meaningful influence can be mapped to a simple author list toward recognizing layers of explicit, statistical, and institutional inheritance. GitHub sits at the boundary between those layers: a place where sources are individually addressable even as their patterns can circulate through machine systems in less individually attributable forms.

What this means for the GitHub paradigm

The training layer completes the argument of the book because it shows why bot-to-bot inheritance can exist even without explicit agent interaction. One machine-assisted repository can influence a later system through direct reading, search indexing, code matching, dataset construction, or learned statistical structure. These channels have different provenance properties, but all make the public artifact part of a machine intellectual environment.

The practical response is not to retreat from public repositories. Publicness remains essential for verification, priority, reuse, and preservation. The response is to make public artifacts more self-describing: stable versions, clear licenses, claim boundaries, provenance summaries, exact references, machine-readable assumptions, and durable links to parents. When explicit attribution is possible, the artifact should make it easy. When influence later becomes diffuse, at least the public record of the artifact itself remains intact.

The repository cannot force future machines to remember where every learned pattern came from. It can ensure that there was something precise to remember. That is a smaller promise than perfect attribution, but it may be the most durable contribution an author can make to a bot-readable future.

Chapter 45 - Scale: When Attribution Costs More Than Generation

The asymmetry between production and evaluation becomes structural.

Generation becomes cheap; understanding does not

The strongest force pushing us toward the end of attribution may be economic rather than philosophical. Generative systems reduce the marginal cost of producing another draft, implementation, proof attempt, review, or repository. The cost of understanding what was produced does not fall at the same rate. A model can generate ten candidate architectures before a human expert can responsibly examine one. An agent can create several pull requests while a maintainer is reading the first. The asymmetry between production and evaluation becomes structural.

Attribution has a cost too. Someone must decide which sources matter, which machine sessions were consequential, which human interventions changed direction, and how much detail belongs in the public record. When artifacts were expensive to produce, a research group could afford to spend substantial effort documenting each one because there were relatively few. When artifacts can be generated continuously, documentation competes with generation for attention. The easiest path is to keep the files and omit the history.

This is why the provenance protocol in this book must be selective. A system that requires one semantic record for every model message will collapse under its own bookkeeping. The important ratio is not provenance events per token but provenance events per consequential transformation. The record should become more abstract as production volume grows. Otherwise the cost of describing the process will exceed the cost of running it.

The repository can grow faster than the reader

Public software already demonstrates a version of this problem. A mature project can accumulate thousands of commits, issues, pull requests, dependencies, and automated updates. No maintainer reads the entire history every time. Git solves the storage problem but not the attention problem. Search, blame, tags, release notes, and tests let people recover the relevant subset. AI-generated research will need analogous compression at the semantic level.

A repository with forty versioned descendants is not automatically more informative than one with four. The additional states matter only if they preserve meaningful differences. In a machine-rich workflow, version numbers can proliferate because revision is cheap. The project therefore needs explicit release gates and family structure so a reader can distinguish exploratory churn

from scientifically important transitions. A patch fixing a parser and a theorem extending primes to prime powers should not occupy the same conceptual level merely because both incremented a version number.

This suggests that semantic releases need hierarchy. Major intellectual events can be summarized at the family level. Minor repairs remain recoverable through Git. Intermediate model sessions can be grouped under a release-level event. A future agent navigating the lineage can request deeper detail only when needed. The provenance graph becomes a multiscale index rather than a flat log.

Credit becomes scarce because attention is scarce

Traditional scholarly credit is valuable partly because human attention is limited. A citation says, among other things, that this prior work deserves notice. When models can generate and discover enormous numbers of related artifacts, the citation economy can become saturated. A future paper might depend mechanically on hundreds of repositories and thousands of machine-generated intermediate objects. Listing every contributor with equal prominence would not inform the reader; it would bury the important dependencies.

The response cannot be simply to stop crediting predecessors. That would reward volume by erasing origin. Instead credit must become structured. Direct conceptual parents deserve prominent citation. Machine-identified supporting artifacts can live in a dependency graph. Routine tool contributions can be recorded in provenance without appearing in the main narrative. Human research direction and release responsibility can remain visible without pretending that all intermediate production was human-authored.

This is similar to software dependencies. A program can depend on hundreds of packages without listing every maintainer on its splash screen. Licenses and package manifests preserve the dependency relationships at the appropriate layer. Scholarship may need a comparable separation between the human-readable argument and the machine-readable contribution graph.

Bot-to-bot production multiplies intermediate artifacts

Bot-to-bot workflows magnify the scale problem because machines naturally communicate through intermediate objects. A planning agent produces a plan. A coding agent turns the plan into a patch. A reviewer produces comments. A fixer produces another patch. A verifier produces reports. A packaging agent produces a release archive. A documentation agent produces release notes. Each artifact can be useful, yet not every artifact should become a permanent scholarly citation target.

The lineage therefore needs garbage collection without historical amnesia. Some intermediate objects can be ephemeral once their consequential effects are summarized. Others should be

retained because they establish why a theorem changed or why a release was rejected. The selection criterion is evidentiary value. Would losing this object make it difficult to reconstruct a major claim, correction, or decision? If not, a summary event may be enough.

This criterion mirrors laboratory practice. Not every temporary file from an instrument is archived forever, but raw data supporting a published result often are. Not every conversation among collaborators is preserved, but a crucial protocol change should be documented. AI does not abolish archival judgment; it increases the volume on which judgment must operate.

The danger of automatic publication

Cheap generation creates a temptation to make publication automatic. If an agent can open a repository and produce a new result, why not tag and release it immediately after tests pass? The answer is that release is not a computational side effect. Publication consumes the attention of future readers, establishes a public priority claim, creates maintenance obligations, and can pollute the literature if the artifact is weak. The lower the production cost, the more important the release gate becomes.

This is especially true for machine-generated mathematics. A system can generate many correct but uninteresting propositions. It can also generate subtly false statements that survive shallow tests. If every candidate becomes a repository and every repository becomes a paper, discovery infrastructure will be overwhelmed. Human or institutionally governed selection becomes a form of literature quality control.

The GitHub paradigm therefore separates artifact creation from public deposit. A machine can produce unlimited local candidates. Only a small fraction should become durable releases. The provenance ledger can record why a candidate crossed that boundary: novelty review completed, independent verification passed, claim boundary narrowed, and a responsible human authorized publication.

Attribution can become algorithmic without becoming fair

One apparent solution to scale is automated attribution. A system could compare new code and prose against public indexes, identify nearest matches, infer repository ancestry, and generate a credit graph automatically. Such tools will be useful, but they will inherit the biases of their indexes and similarity measures. Exact or near-exact copying is easier to detect than abstract influence. Famous repositories may be overrepresented. Deleted, private, paywalled, or non-English sources may be missed.

Automated attribution should therefore be treated as evidence rather than judgment. A code-reference match is a strong clue about a particular textual relationship. A semantic similarity score is a weaker clue about intellectual ancestry. A literature agent's claim that two theorems are

equivalent requires mathematical review. The more abstract the relation, the more the system should expose uncertainty.

Fair credit will remain partly normative because importance is not the same as similarity. A short conceptual note can matter more than a thousand copied lines. A reviewer who discovers a fatal counterexample can matter more than the model that generated the original proof. Algorithms can help recover relations; communities still decide which relations deserve recognition.

The provenance budget

A useful practical concept is a provenance budget. Every project has limited time for documenting origin. Spend the budget where later uncertainty would be expensive. Record the birth of the core idea, major changes in claim scope, discovery of prior art, independent verification pathways, negative results that terminate branches, and the human decision to release. Let ordinary Git history handle routine implementation detail.

The budget can be increased automatically where tools already generate trustworthy metadata. CI runs can emit verification events. Agent platforms can record session identifiers. Code-reference systems can attach source matches. Release tooling can capture hashes. Automation should reduce the human cost of provenance, leaving human attention for semantic events that machines cannot identify reliably.

This principle keeps the proposed system realistic. The goal is not a museum-quality archive of every thought. It is enough structured memory that another researcher can understand why the artifact exists, what it inherited, how it was challenged, and who stands behind it.

Scale changes the meaning of the end of attribution

At small scale, attribution fails because we do not know who did what. At large scale, attribution can fail even when the information exists because there is too much of it to be useful in flat form. A project may possess exact logs for hundreds of agents and still need to compress them into a meaningful story. The end of attribution is therefore partly an information-architecture problem. The old representation - a short list of authors - has too little capacity, while the raw event stream has too much.

The contribution graph occupies the middle. It compresses thousands of operations into a few dozen causal events. It can preserve direct ancestry without pretending that every influence is recoverable. It can keep human responsibility prominent while acknowledging machine production. It can scale from a public one-page summary to a deep machine-readable archive.

This is the same solution Git brought to file history: do not preserve only the latest file and do not force the user to read every byte of every version. Preserve a structured graph and provide

ways to traverse it at the needed resolution. The GitHub paradigm extends that idea from changes in files to changes in intellectual state.

PART VIII - THE REPOSITORY AS HABITAT

Chapter 46 - The End of Attribution?

A byline, a commit author, or a repository owner can remain true while becoming radically incomplete.

An ending, not an erasure

The phrase 'end of attribution' is intentionally provocative, but the argument is now precise. Names will not disappear. Legal authorship will not disappear. Credit will not disappear. Commit attribution will not disappear. What ends is the adequacy of attribution as a complete description of intellectual origin. A byline, a commit author, or a repository owner can remain true while becoming radically incomplete.

The replacement is not anonymity. It is structured provenance. We should know which artifacts descended from which, which actors played which roles, which reviews changed which claims, which systems generated candidates, which verifiers certified results, and who accepted responsibility for release. This is a richer record than conventional authorship, not a poorer one.

GitHub is the emblem because it already teaches us to think in graphs. There is no single final file without history; there is a lineage of states. The next conceptual move is to accept that ideas also have lineages, and that their lineages can cross humans, models, tools, and repositories without respecting the boundaries of a byline.

Who thought of it?

The oldest question in attribution is 'who thought of it?' In simple cases the answer remains meaningful. In complex human-machine systems it may not. A result can emerge from a human objective, a model's unexpected construction, a second model's objection, a deterministic verifier's failure, and a human decision to reinterpret the failure. Removing any one event might remove the final result. Origin is distributed.

Science has encountered distributed causation before. Modern experiments depend on instruments, software stacks, collaborations, prior datasets, and institutional infrastructure. AI makes the distribution enter the generative core of reasoning itself. The artifact can be produced by a network whose human participants understand the goal and whose machine participants supply unanticipated transformations. The honest history may therefore be a graph rather than a name.

That does not diminish human significance. It changes where human significance is located. Direction, judgment, values, risk acceptance, and the decision to publish become more visible once we stop measuring authorship by keystrokes. Machine contributions also become more

visible once we stop hiding them behind the uploader's account. Both forms of visibility are improvements.

The GitHub paradigm is ultimately a proposal to take our own technical infrastructure seriously. We already know how to preserve ancestry, compare descendants, certify artifacts, and inspect transformations. We now need to apply the same seriousness to the ancestry of ideas. If we do, the end of attribution may become the beginning of a more accurate history of thought.

Further implications

The argument therefore ends with a practical rather than metaphysical recommendation. Keep the byline. Keep commit authors. Keep legal responsibility. But stop asking those fields to carry the entire history. Add a contribution graph that can represent human direction, model generation, bot review, deterministic verification, semantic descent, and release decisions. The richer record makes old forms of attribution more trustworthy because it clarifies what they do and do not mean.

If that practice becomes normal, the title's final phrase may sound less apocalyptic. 'The end of attribution' would mean the end of a historical shortcut: one artifact, one originator, one name. In its place would be a more faithful account of collaborative cognition, one that can survive the transition from human teams to mixed networks of humans, models, and machines.

Appendix A - Repository Corpus Snapshot

This appendix records the public repository corpus visible from the connected GitHub account during preparation of this manuscript on August 30, 2026. Repository names are preserved as research objects rather than normalized into a bibliography. Several names are version-specific repositories and therefore show the historical practice of depositing stable release states as separate public objects.

The corpus is important not because every repository is equally central to the argument, but because the list makes the branching pattern visible. There are base projects, explicit version descendants, viewers, application ports, research engines, mathematical families, controls, and small proof-of-principle tools. The collection is a snapshot, not a claim that no repositories existed before or after the access date.

The snapshot contains 56 repository names visible to the connected account at the time of inspection. The names themselves show several explicit lineages.

1. abstract-algebra-calculator
2. algebraic-abiogenesis
3. algebraic-abiogenesis-stage2
4. alien-invariance
5. cellular-algebraic-abiogenesis
6. computational-causal-geometry
7. computational-causal-geometry-v0.3.2
8. computational-causal-geometry-v0.4.1
9. computational-causal-geometry-v0.5.1
10. computational-causal-geometry-v0.6.2
11. cycle-holonomy-tomography
12. determinant-dual-obstruction-family
13. determinant-dual-obstruction-family-v0.3.1
14. doppelganger-worlds
15. fractal-state-space
16. fractal-state-space-v0.3.1
17. legos-relation-before-object
18. locality-causality-fork
19. mathematics-of-the-genetic-code-in-a-python-program
20. order-disorder-lab
21. order-disorder-lab-v0.3.1

22. projective-dual-obstruction
23. relational-constraint-dynamics
24. relativistic-observer-complexity
25. relativistic-observer-complexity-v0.3.1
26. relativistic-observer-complexity-v0.4.1
27. state-space-doors
28. state-space-doors-v0.2.2
29. state-space-doors-v0.3.2
30. state-space-doors-v1.0.1
31. state-space-doors-v1.1.0
32. stereo-disparity-topology
33. stereo-induced-unreal-cube
34. stereo-multistability-android
35. stereo-multistability-atlas
36. stereo-sketch-3d
37. theory-preserving-ontogenesis
38. theory-preserving-ontogenesis-v0.8.0-relation-view-degree-
39. theory-preserving-ontogenesis-v1.0.0-spark-degree-law-
40. theory-preserving-ontogenesis-v1.0.1-divine-spark-complexity-law
41. theory-preserving-ontogenesis-v1.1.1-self-indexing-seed-
42. theory-preserving-ontogenesis-v2.0.1-seed-univalent-completion
43. theory-preserving-ontogenesis-v3.2.3-runtime-bound-fail-closed-model-drift-bond
44. topological-response-renormalization-
45. tuple-imagery-observatory-v0.1.0-
46. youvan-ai-v0.3.0-direct-stratified-world-transfer-
47. youvan-ai-v0.34.0-minimal-seed-multiscale-birth-rank-navigation-index-
48. youvan-ai-v0.55.0-auditable-theorem-workflow-composition-
49. youvan-ai-v0.72.2-certified-residual-error-anonymous-local-gate-
50. youvan-ai-v1.0.0-certified-ternary-ontogenesis-
51. youvan-ai-viewer-v0.1.0-local-discovery-observatory-
52. youvan-ai-viewer-v0.2.0-evolving-math-observatory-
53. zero-learning-chatbot-
54. zero-learning-chatbot-katerina-tikhonova-book-edition
55. zero-learning-chatbot-katerina-tikhonova-book-edition-v0.2.1
56. zero-learning-chatbot-katerina-two-corpus-v0.3.1

Appendix B - Minimum Viable Provenance Schema

A minimal provenance ledger can be stored as newline-delimited JSON. The following fields are intentionally conservative. They describe observable project events rather than hidden reasoning.

schema_version: Version of the provenance-event schema.
event_id: Stable identifier for the event.
timestamp: Time the event was recorded or occurred.
project: Repository or research-object identifier.
input_artifacts: Parent commits, releases, files, datasets, or documents materially used.
actor_type: human, generative_model_session, deterministic_tool, ci_service, organization, external_reviewer, or unknown.
actor_id: Public identifier when appropriate; may be redacted or pseudonymous.
model: Provider/model/version information for generative sessions when known.
role: conceived, generated_candidate, implemented, reviewed, falsified, verified, selected, rejected, generalized, edited, released, maintained, or another controlled term.
summary: Short description of the consequential change.
output_artifacts: Commits, patches, issues, result files, releases, or documents produced.
evidence: Optional links to review reports, test runs, issue comments, or signed attestations.
confidence: Optional declaration when historical reconstruction is uncertain.
responsibility_owner: Human or institution accepting release responsibility when the event is a release.
signature: Optional cryptographic signature or attestation reference.

A single project can expose a public summary ledger while retaining more detailed private logs. The public ledger should prioritize events that changed claims, architecture, evidence, scope, or release status. Routine formatting and mechanical refactoring can remain in ordinary Git history.

Example event, expressed conceptually rather than as a binding standard:

```
{
  "event_id": "evt-0042",
  "project": "determinant-dual-obstruction-family",
  "actor_type": "generative_model_session",
  "model": "model-family/version",
  "role": "generalized",
  "input_artifacts": ["projective-dual-obstruction@v0.1.1"],
  "summary": "Proposed extension from prime fields to odd prime-power fields
and identified the q=9 generation issue as requiring explicit treatment.",
  "output_artifacts": ["draft theorem and proof plan"],
```

```
  "confidence": "declared by project maintainer"
}
```

The important property is not the exact field names. It is the separation of roles. A downstream reader can see that generation, review, verification, selection, and release are different events performed by potentially different actor types.

Appendix C - Practical Adoption Checklist

Projects can adopt intellectual provenance without waiting for a platform-level standard. A practical workflow is:

1. Keep conventional Git history clean and meaningful. Do not replace commits with provenance metadata.
2. Add a PROVENANCE.md or PROVENANCE.jsonl file at the repository root.
3. Record the model family and date when generative systems materially create code, proofs, prose, analyses, or review findings.
4. Record consequential human steering acts: rejecting a release, narrowing a claim, changing architecture, or selecting a new research direction.
5. Treat independent AI reviews as review events, especially when they cause changes.
6. Link provenance events to exact commits or release tags.
7. Preserve claim boundaries and limitations in explicit files.
8. Distinguish reproduction, verification, and interpretation.
9. Identify a human or institution responsible for each public release.
10. Avoid publishing private chain-of-thought as a provenance requirement. Record decisions and evidence instead.
11. Use stable release identifiers and hashes when a descendant project depends on a specific parent state.
12. When a new repository is conceptually descended from another without sharing code, declare a semantic parent relation rather than relying on a Git fork.
13. When prior art changes the novelty claim, record that change as a provenance event.
14. When a version is intentionally not deposited, preserve the reason in local project history if it materially explains the next release.
15. Periodically summarize the event ledger so the public history remains readable.

A future GitHub interface could automate much of this. Agent sessions could emit signed contribution events. Pull-request reviews could become typed provenance nodes. Releases could include responsibility attestations. Semantic parent relationships could complement forks. Until then, a small text or JSON file is enough to begin preserving the history that ordinary commits cannot express.

Appendix D - Thirty Questions for a Provenance Audit

A provenance audit is not a philosophical exercise. It can be performed as a practical release review. The following questions are designed for a research repository that uses generative AI materially.

1. What exact artifact is being released: commit, tag, archive, package, paper, dataset, or combination?
2. What earlier artifact is its immediate technical parent?
3. Does the release have a conceptual parent that is not a Git parent or fork?
4. Which human set the principal research objective?
5. Which generative systems materially produced code, mathematics, prose, data analysis, or figures?
6. Were any machine sessions used as independent or firewalled reviewers?
7. Did a review cause a claim, theorem, architecture, or scope to change?
8. Which deterministic tools or verifiers supplied evidence used in the final claim?
9. Were independent implementations genuinely independent, or did they import common code?
10. What assumptions are supplied rather than derived?
11. Are those assumptions machine-readable where feasible?
12. Which results are intrinsic and which depend on coordinate, seed, tie-breaking, or runtime conventions?
13. What prior art was found during development, and did it change the novelty boundary?
14. Are classical ingredients separated from candidate-new synthesis?
15. Were any versions intentionally withheld from public deposit?
16. If so, what defect or uncertainty caused withholding?
17. Did a negative result terminate a branch or motivate a new project?
18. Are such semantic parent-child relations recorded explicitly?
19. Can the principal computational claims be reproduced from the released artifact without access to private chat history?
20. Can another model or human reviewer understand the limitations without inheriting the developer's advocacy context?
21. Are all important result files covered by a manifest or equivalent integrity check?
22. Does the release distinguish reproduction from independent verification?

23. Does it distinguish exact same-runtime identity from scientifically acceptable cross-runtime agreement when relevant?
24. Who accepts responsibility for correction, maintenance, or retraction of the release?
25. Is that responsibility distinguishable from who generated the content?
26. Are model identifiers recorded at a useful level without exposing unnecessary private data?
27. Are critical contribution events preserved without publishing private chain-of-thought?
28. Can a future descendant cite the exact release rather than only the repository name?
29. Could a historian reconstruct why the final claim has its present scope?
30. Could a machine agent reconstruct the same lineage automatically?

A project need not answer every question perfectly to benefit from the exercise. The audit is intended to reveal where ordinary Git history is sufficient and where semantic provenance is missing. The most valuable answers are often the ones that expose a gap before publication, such as discovering that an 'independent' verifier imports the same helper as the primary implementation or that a conceptual parent is mentioned nowhere in the release.

The audit also creates a disciplined boundary around AI disclosure. Instead of asking the vague question 'how much AI was used?', it asks what the AI did, what evidence survived, and what decisions changed because of it. That is both more informative and more resistant to performative disclosure.

References and Source Notes

The book is primarily an argument grounded in repository practice. The following sources support the descriptions of current GitHub agent features, attribution mechanisms, software-provenance systems, and U.S. copyright policy. Repository case studies were read from their public README and release files during preparation.

GitHub Docs. 'Creating a commit with multiple authors or on behalf of an organization.' Accessed August 30, 2026.

GitHub Docs. 'About GitHub Copilot code review.' Accessed August 30, 2026.

GitHub Docs. 'GitHub Copilot code referencing' and 'Finding public code that matches GitHub Copilot suggestions.' Accessed August 30, 2026.

GitHub Docs. 'About third-party coding agents.' Accessed August 30, 2026.

GitHub Docs. 'Use Copilot agents.' Accessed August 30, 2026.

SLSA. 'Provenance,' Specification v1.2. Accessed August 30, 2026.

in-toto. 'Attestation Framework' and in-toto specification documentation. Accessed August 30, 2026.

Sigstore. 'Overview' and 'Rekor' transparency-log documentation. Accessed August 30, 2026.

U.S. Copyright Office. Copyright and Artificial Intelligence, Part 2: Copyrightability. January 2025.

Youvan, Douglas C. GitHub public repositories, repository README files and release documentation examined August 30, 2026, including Theory-Preserving Ontogenesis, Youvan.ai, State-Space Doors, Relational Constraint Dynamics, Determinant-Dual Obstruction Family, Stereo-Induced Unreal Cube, Abstract Algebra Calculator, and Zero Learning Chatbot.

Chacon, Scott, and Ben Straub. Pro Git, 2nd ed. Apress; freely available through git-scm.com.