Decision Support for Pre-clinical Drug Discovery Without Agentic Engineering: One Pinned Reasoning Engine and a Human Who Decides

Raminderpal Singh
Independent Researcher
raminderpal@hitchhikersai.org
Preprint. September 23, 2026.

Abstract. Agentic engineering, coding agents that read the repository and execute changes under automated checks, and products built as teams of specialist agents, is the prevailing direction of AI-assisted software development. This paper argues that for decision support in pre-clinical drug discovery the prevailing direction is right about process and wrong about architecture. In that setting a scientist or a committee acts on the software's output, the reasoning engine is stochastic, one accountable person must defend every mechanism, the record must survive into publication, and the science and data decisions that shape the system are forced by what the sources hold, which is discovered only by building and testing against them. These divide into requirements on the output at run time and requirements on the method at design, development and test time, and we treat them separately. We describe a build method and a product architecture developed for two independent systems, Compound Insights and the SIM Framework, and compare both with agentic engineering practice and many-agent architectures using the 2025-2026 empirical literature. The build method keeps the process claim of agentic engineering, external checks, a human holding judgment, failures turned into standing controls, and differs in one respect: the operator executes the deterministic layer. The product architecture uses one pinned reasoning engine, a provenance record on every value, replication in place of consensus, and no verdict emission. We state what the approach costs and give four questions that follow from the properties and apply to any decision-support system in this setting. The comparisons are design assessments against a stated standard, not controlled measurements.

Keywords: agentic engineering; multi-agent systems; provenance; reproducibility; decision support; drug discovery; development candidate; large language models; human-in-the-loop

1. Introduction

Two developments define AI-assisted software engineering in 2026: the coding agent, a model with read and write access to the repository, executing commands under hooks that block rule violations and organized as a harness that persists state across context windows [15,16]; and the many-agent product, an orchestrator that dispatches planner, researcher, critic and writer agents and merges their outputs [1,17]. The empirical record behind them is mixed. A randomized trial found experienced developers 19% slower with early-2025 tools while believing they were faster [2]. A taxonomy of multi-agent failures found that gains over single-agent frameworks are often minimal and that failures cluster in specification, inter-agent misalignment and weak verification [1]. Two 2026 case studies agree that the process, whichever tool is used, determines whether AI-assisted development is governable: external checks, a human holding judgment, and failures converted into durable controls [3,4].

This paper takes that process claim as correct and asks what follows when the software is decision support for pre-clinical drug discovery, where a scientist or a committee acts on the output. We report a build method and a product architecture developed over two independent systems, Compound Insights, a drug-compound dossier engine, and the SIM Framework, decision support for the Development Candidate (DC) decision. They have separate scopes, codebases and deployments; they share the method, the reasoning engine each deploys, and the properties in Section 2. Section 3 states what agentic engineering claims; Sections 4 and 5 give the method and the architecture with a comparison table each; Section 6 catalogs observed failures and their controls; Section 7 describes how each system applies the approach and why the conventional build was rejected; Section 8 states the costs and the limits of the evidence.

2. Setting and position

Pre-clinical drug discovery is a sequence of decisions taken on incomplete evidence: which target to pursue, what the public record says about a compound, whether a series is worth optimizing, and, at the DC decision, whether to nominate one molecule and commit it to IND-enabling toxicology. Analyses of attrition link late failure to the quality of evidence and decisions at these earlier points [12], and the long-run decline in R&D efficiency to systematic causes in how projects are chosen and progressed [11]. Each decision is contested, and a person or a committee resolves it. Decision support here has one job, to assemble the evidence for and against each claim with its provenance visible, and one prohibition, never to make the decision itself.

Two sets of requirements follow, and they are distinct. The first concerns the output at run time, which is what makes it useful to the person acting on it. R1, consequential: a scientist or a committee acts on the output, so a plausible wrong answer costs more than a slow right one. R2, attributable: every value traces to the evidence and the engine version that produced it. R3, reproducible: the reasoning engine is a language model whose outputs vary between runs [18], so a value is trusted only when replicated under a pinned version, with the spread published. R4, undecided: the system assembles the evidence and does not make the decision; disagreement is shown, not resolved.

The second set concerns how the system is designed, developed and tested. M1, accountable: one person must be able to defend every mechanism to a reviewer, a co-author or a partner. M2, recorded: falsified hypotheses and measured failures are results and must survive into publication. M3, externally verified: because the engine is stochastic and a model cannot check its own output, every change is verified outside the model and every prediction is scored. M4, decided by a scientist as the data surfaces the decision: the science and data decisions that shape the system cannot be specified in advance. Pre-clinical evidence is sparse and uneven, and what the public record holds for a given compound, target or precedent is discovered only when the system is run against the sources; each gap forces a decision that is scientific, not technical. A data decision (which source, which field) and a coding decision (which fallback) can be delegated with a rule; a science decision turns on what the evidence supports, is argued before it is made, and needs a person who can be held to it.

The build method of Section 4 is designed to meet M1 to M4; the product architecture of Section 5 is designed to meet R1 to R4. One discipline serves both: the register that satisfies M2 is the same habit as the provenance record that satisfies R2, and the replication campaign that verifies a change (M3) is the same instrument that establishes a value (R3). Agentic engineering, as Section 3 shows, addresses M3 well and M1, M2 and M4 partly; the many-agent architecture meets R2 to R4 only in part, with provenance per agent, no pinned replication, and a converged answer where R4 asks for a decision left open.

The position of this paper, in plain terms. Build what the decision needs and nothing beyond it. Put a human at every point where judgment is exercised, and let the human decide. Decide the architecture consciously, as a fixed core with layers built on it, rather than letting a general agent design its own workflow at run time. Where agents choose their own path, what happens is partly random; that is sometimes presented as a feature, and in this scope it is a defect, because even when the agents reach a right answer the path to it is not clean on provenance or on who decided, and those have to be put in by design.

Compound Insights [21] answers what the public record says about a compound: cited dossiers from ChEMBL [13], UniProt, PubMed, ClinicalTrials.gov, Open Targets and Reactome plus literature-grounded inference, under the principle attribute, never assert, read by scientists. The SIM Framework [24] supports the DC decision: for each claim in a nomination package, the case that the evidence is sufficient, the case that it is not, and an adjudication, repeated several times with the disagreement reported rather than averaged, read by a committee; its published demonstration argues a synthetic candidate, an oral NLRP3 inhibitor for Alzheimer's disease, against real public precedents. The two share no code or files. Each deploys, as a pinned dependency, the same reasoning engine, project fred [22], version 0.3.0: a ReAct-style lookup loop [5] that records every lookup.

3. What agentic engineering claims

The tooling is a coding agent with repository access, hooks that block rule violations, and a harness that carries state across context windows through progress files, feature lists and git history [15,16]. The process claim is separable from the tooling. Moreira [3] states it directly: absent external tool use, no language model can determine whether what it generated is correct, so verification must be external to the

generator and enforced at approval steps the agent may request but not pass. Davis et al. [4] add, from a 12-week instrumented case, that velocity surfaces recurring failure classes and that engineering judgment sustains velocity by converting them into durable governance mechanisms. The METR trial [2] supplies the base rate: developers' beliefs about their own speed-up were uncorrelated with the measured effect. Models do not reliably correct their own reasoning without external feedback [6], and model-based judges show position and self-enhancement biases and favor their own generations [7,8].

The many-agent product decomposes a task across role agents that pass state and converge on an answer. MAST [1] analyzed more than 1,600 execution traces across seven frameworks and found fourteen failure modes in three categories: specification and system design, inter-agent misalignment, and task verification; gains over single-agent baselines were often minimal. Multi-agent debate improves factuality on some tasks [9], but debate strategies do not reliably outperform self-consistency or ensembling and are sensitive to their agreement settings [10]. Anthropic's multi-agent research system [17] places the benefit in fan-out over independent subtasks and the cost in coordination and token use. Many-agent designs gain parallelism and breadth at the cost of coordination failures that are hard to attribute afterwards.

4. The build method

The method was developed by one operator, a systems engineer by training and not a professional developer, building both systems in interactive sessions with a consumer LLM application (Claude.ai) and a terminal, with no coding agent and no repository access for the model [20]. The operator makes every science and governance decision; the model writes every line of code, every document and every check. Figure 1 summarizes the loop; five elements follow, and a sixth paragraph says why they form a loop.

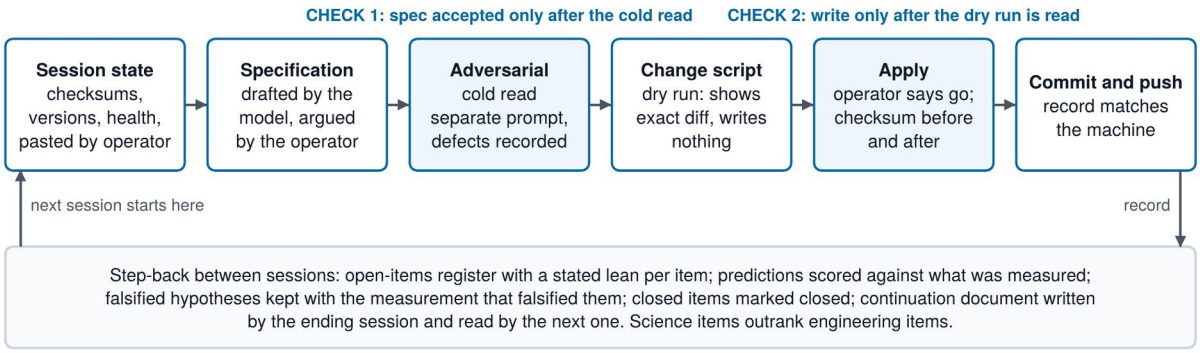

Figure 1. The build loop. Two checks (shaded) sit between drafting and writing: a specification is accepted only after an adversarial cold read, and a write happens only after the operator reads the dry run. The step-back band is the record that carries state between sessions.

Standing instructions. Behavior is set in four layers: personal settings for every session, project instructions for one application, a continuation document written by the previous session, and the conversation. The first two are plain-English files. When behavior drifts the fix goes into one of them, not into the next message. Harness designs make the same move with progress files [15]; here the operator, not a harness, reads and enforces them.

Specification before code, attacked before acceptance. The model drafts a specification from a conversation in which it asks and the operator decides. A separate prompt written to find fault reads it cold; defects are fixed and the withdrawn claims stay in as a record; only then is code written. On one Compound Insights specification that read found eleven defects, five structural, before any code existed [20].

Checked change scripts. Every change is a script the model wrote and the operator ran. It states the checksum it expects, shows in a dry run exactly what it would change, and aborts if the file is not as expected. The operator reads the dry run and says apply; checksums confirm the result; the change is committed and pushed. One script aborted correctly on finding four matches where it expected two; the extra two were historical record [20].

Predictions scored, misses kept. The expected outcome of a change is written down before it runs and scored afterwards. Misses are the material from which controls are built (Section 6); this is what Davis et al. call governance conversion [4], done by hand.

The step-back. Between sessions the operator forces the review a session does not produce on its own: an open-items register with a stated lean per item, science items above engineering items, closed items marked closed, and a continuation document. Left alone, a project drifts toward whatever is tractable; the step-back re-asserts priority.

Decisions surfaced by the data. M4 is why the method is a loop rather than a specification followed by a build. A run against the real sources reveals what they hold: a compound with no ChEMBL entry and three PubMed records; a claim for which no evidence exists in any source; a precedent whose public reference names no parseable accession. Each finding forces a ruling that no specification anticipated: whether the dossier reports no data or infers from analogues; which sources count as curated and which fields permit literature-grounded inference; what comparability basis a precedent must carry; whether the engine declining is the correct result. Running a language model at inference time adds a second layer, because its behavior on thin data, whether it fills, declines or over-generalizes, is measured, not predicted. The loop is therefore: run, find the gap, put the decision to the scientist, record the ruling in the register before it takes effect, update the specification, run again. A coding agent resolves such decisions as it goes, choosing a fallback, and the choice is visible only if it is logged; a many-agent product resolves them at run time, differently on each run, where no one sees them. A harness can be instructed to stop and ask at a data gap, but only at the gaps its author anticipated, and what it returns is a yes-or-no prompt. A science decision is not answered that way. Whether a compound with three PubMed records should carry any inferred claim at all, or what makes a twelve-week trial comparable to the chronic exposure a candidate proposes, is settled by discussion: the operator arguing it with the model, and outside the session with co-authors and domain reviewers, before a ruling is written down that someone will defend. The method keeps that discussion where it happens, in the session and the register, and keeps the person who made the ruling attached to it.

Table 1 compares the method with coding-agent practice. The assessments are the operator's judgment against the projects' standard (attribute, never assert; no silent failure; measured, not inferred) and the cited literature, not controlled measurements. The rows that favor agentic tooling matter as much as the others: the method is not claimed to be faster, to read the live repository, or to transfer.

Table 1. The build method against agentic engineering practice.

Dimension	This method (interactive sessions; operator executes)	Agentic engineering (coding agent with repository access and hooks)	Assessment
Locus of scientific and governance decisions	Operator, one ruling at a time, recorded before it takes effect	Engineer at review points; agent proposes. With agent teams, judgment is spread across role prompts	Equal with a single agent; this method against many
Science and data decisions surfaced during build (M4)	Surfaced by a run against the sources; ruled by the operator; recorded before taking effect; specification updated	Resolved by the agent as it goes, visible only if logged; with agent teams, resolved at run time and differently per run	This method
Verification that a change is correct	Every write checked: checksum, anchor count, self-tests, scratch render; operator approves	Same checks expressed as hooks that block the action; many-agent systems often satisfy them vacuously [1]	Equal with a single agent; this method against many
Model reads the live repository	No; it reads uploads. Source of the most frequent build-time error (Section 6)	Yes, from disk	Agentic tooling
Execution of commands	Operator types, runs and pastes every block	Agent runs; engineer approves	Agentic tooling on time; this method on comprehension

Dimension	This method (interactive sessions; operator executes)	Agentic engineering (coding agent with repository access and hooks)	Assessment
Record of belief versus measurement	Every prediction scored; every miss and every falsified hypothesis kept with its measurement	Discarded at context compaction unless the harness persists it [15,16]	This method
Silent failures during the build	Rare within a session; occur at session boundaries; caught by continuation records	Rare if every hook fails loudly; common otherwise	Equal; set by discipline
Throughput	Bounded by operator turn rate; findings outrun triage	Higher on execution; evaluation campaigns remain serial because the engine runs one job at a time	Agentic tooling
Operator's understanding of the system	High; every line and every result is read	Erodes unless transcripts, not summaries, are read	This method
Transfer to a second engineer	Not possible today; rules are prose	Possible where rules are hooks	Agentic tooling

On who decides, how a change is verified and what is recorded, the method and the tooling implement the same process. On who executes the deterministic layer they differ, and that costs the method learning speed and transfer, not correctness. On the decisions the data forces during the build, the method differs by design: it puts each one to a scientist as it arises.

5. The product architecture

5.1 One pinned reasoning engine

Each product calls one reasoning engine, project fred, deployed separately and treated as a frozen dependency: pinned to a version, called through one path, never edited by the application. fred is a ReAct-style loop [5] (thought, one lookup against a public source or the literature, observation, repeat within a step budget) that records every lookup: source, query, time, duration, found or not, and a fingerprint of the reply. Recall can fabricate; a recorded lookup can be checked by a reviewer without trusting the operator or the model. An earlier Compound Insights design used a general agent framework; it was retired for a single engine with one measured call path, so that no framework-level silent failure remains. Figure 2 contrasts the two arrangements.

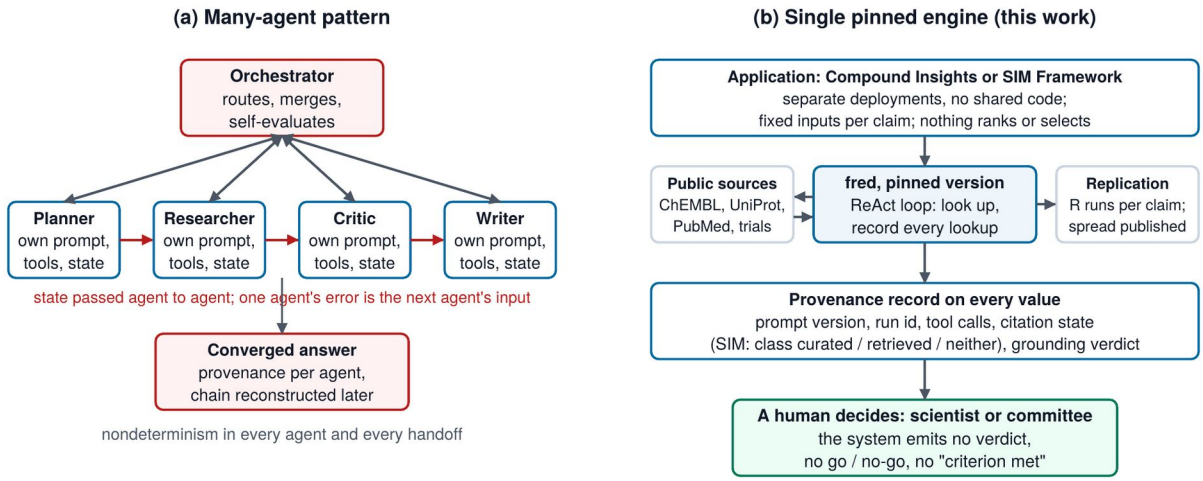

Figure 2. Product architecture. (a) The many-agent pattern: an orchestrator dispatches role agents that pass state to one another; nondeterminism lives in every agent and every handoff, and provenance is reconstructed afterwards. (b) This

work, deployed separately in each system: one pinned engine behind fixed inputs, lookups recorded, replication published as spread, a provenance record on every value, and a human who decides.

5.2 Provenance on every value

Every inferred value in a Compound Insights dossier carries the prompt version, run identifier, lookup record, per-citation validation state on independent re-resolution, and a grounding verdict. The SIM Framework classes every citation row by origin: curated, resolving to the corpus supplied to the engine; retrieved, evidenced by a recorded tool result; or neither, an identifier in no supplied record and no recorded search. The third class is the safety-relevant one and is published rather than suppressed [24]; fabricated and erroneous citations from language models are documented [28], which is why every citation is re-resolved and, in SIM, classed. A claim with no evidence is reported as such; for a compound with three PubMed records and no ChEMBL entry, most fields read "no data found", and whether the engine declines where it should is itself a test [20]. This applies provenance practice in computational science [14,19] value by value.

5.3 Replication in place of consensus

Because the engine is stochastic (R3), a value is trusted only after repeated runs against a pinned version. Compound Insights runs evaluation campaigns that compare answers to exact values a database holds today, re-resolve every citation, score answers with a second model calibrated on deliberately corrupted answers, and compare against a stored baseline that refuses to run if it has itself moved. SIM argues every claim R times and publishes the spread. Both use self-consistency [23] for measurement, not answer selection: the spread is published, not collapsed by majority vote. An engine change is a deliberate re-pin recorded per claim, and legacy outputs are never back-filled; both systems now run fred v0.3.0.

5.4 Three fixed roles and a human committee

SIM runs the one engine in three fixed roles per claim: a constructive opening that argues the claim from a fixed evidence menu, a skeptical opening that argues against it from the same menu, and an arbiter that reads both, checks citations, records a verdict and may decline to rule. The menu is identical for both arms on every repetition; nothing ranks or selects. The spread is published rather than resolved, so a minority position stays visible; debate protocols that merge positions do not reliably outperform simpler sampling and depend on their agreement settings [10]. A repetition with no verdict is counted, not dropped. A precedent may be cited only with a written comparability basis, and negative-precedent records state what the literature does not contain, since a corpus of papers cannot carry the proposition that no paper exists. The system never emits go, no-go, or "criterion met"; the committee decides (Figure 3).

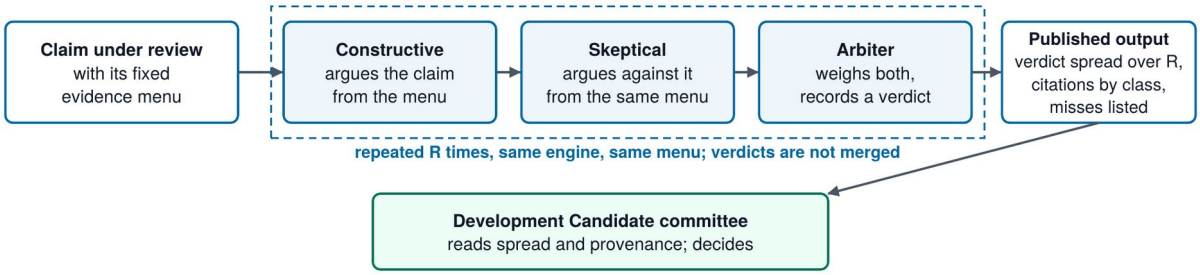

Figure 3. The SIM Framework per-claim pipeline. The same engine runs three fixed roles over the same fixed menu, R times; verdicts are published as a spread, not merged; the committee decides.

Table 2 compares this architecture with the many-agent pattern, on the same basis as Table 1.

Table 2. The product architecture against the many-agent pattern.

Dimension	This architecture (one pinned reasoning engine)	Many-agent architecture (orchestrator plus role agents)	Assessment
Number and kind of agents	One engine; in SIM it runs three fixed roles (constructive, skeptical, arbiter)	Orchestrator plus five to ten role agents	This architecture: fewer places for unattributed inference

Dimension	This architecture (one pinned reasoning engine)	Many-agent architecture (orchestrator plus role agents)	Assessment
Where nondeterminism lives	One place, measured by replication campaigns	Every agent and every handoff	This architecture
Attribution of a value to evidence	Prompt version, run id, tool calls, citation state and grounding verdict on every value; in SIM every citation row classed curated, retrieved or neither	Per agent; the cross-agent chain is reconstructed after the fact	This architecture
Reproducibility	Claimable: engine pinned, campaigns replicated, re-pins recorded per claim, legacy never back-filled	Rarely claimable; inter-agent state drifts; engine changes are config edits without output attribution	This architecture
Who decides	Human committee; no go/no-go, no "criterion met"	System converges and often self-evaluates [1,7,8]	This architecture
Disagreement	Kept and published as verdict spread across repetitions; a missing verdict is counted, not dropped	Resolved in the loop; debate protocols merge positions and are sensitive to their agreement settings [10]	This architecture
Silent failure	Forbidden by rule; detected and logged where unavoidable	Common; one agent's error becomes the next agent's input [1]	This architecture
Throughput per claim	Serial: one engine, one run at a time	Parallel fan-out over independent subtasks	Many-agent, on speed
Breadth of task	Narrow by design: literature review and claim inference	Broad: research, synthesis, writing, tool use in one pipeline	Many-agent, on breadth
Failure classes	Mechanical and testable: stalled runs, serialization, contamination on answer shape	Specification ambiguity, inter-agent misalignment, weak verification [1]	This architecture: failures are visible

The two rows that favor the many-agent pattern, throughput and breadth, are real and do not apply here: the engine serializes runs, so fan-out has nothing to parallelize, and the task is narrow by design because breadth multiplies unattributed steps. The rows that favor this architecture are what R1 to R4 require.

6. Observed failures and the controls that catch them

Tables 3 and 4 list failures observed in the two project records, the mechanism behind each, and the control that now catches it; each run-time row names the system, since a control in one is not a control in the other. Most were added after the failure was observed and scored, as Section 4 describes. In MAST's terms [1], the run-time failures are task-verification failures made visible by a check that can go red, and the build-time failures are specification and verification failures of a single agent, with no inter-agent misalignment because there is one agent.

Table 3. Build-time failures observed in the Compound Insights build record and their controls.

Build-time failure (observed)	Mechanism	Control that catches it
Model reasons from a remembered copy of a file, not the live one	Uploads are snapshots; the live file moves; the model edits what it remembers	Checksum before and after every write; abort on mismatch
Model reports a review that was only a re-read	Checking while writing is not a review; self-verification is unreliable [6]	Review is a separate pass with a separate fault-finding prompt, run before the operator reads

Build-time failure (observed)	Mechanism	Control that catches it
Model announces a deliverable that is not there	Fluent completion text is uncorrelated with the artifact existing	Operator looks for the file card and checksum, not the sentence
Task quietly narrows or widens	Adjacent defects are more tractable than the objective; the model drifts to them	Objective stated at the top of every reply; advanced or not advanced recorded
Change script would touch historical record	A pattern matches more places than intended	Script asserts the expected match count and aborts otherwise

Table 4. Run-time failures observed in the two systems and their controls.

Run-time failure (observed)	Mechanism	Control that catches it
Stalled (zombie) run (Compound Insights)	Engine job neither completes nor fails	Poll limit; the run is recorded as failed rather than dropped, under the no-silent-failure rule
Contamination on answer shape (Compound Insights)	Judge or downstream step keys on format, not content	Second model calibrated on deliberately corrupted answers; cell-by-cell comparison to a frozen baseline
Genuine citation, wrong entity (engine harness, Compound Insights)	A resolvable identifier that is not the entity asked for (Section 6)	Every citation re-resolved independently; run refused before judging
Precedent used as unearned support (SIM)	A prior compound's outcome cited without a stated comparability basis	Negative-precedent records; explicit comparability basis per precedent
Arbitration returns no verdict (SIM)	The arbitration stage completes without a verdict, cited set or reasoning	The claim's no-verdict counter records the loss; the gap is published as a limitation on the provenance surface
Baseline drift (Compound Insights)	The yardstick moves between campaigns	Comparison refuses to run if the stored baseline checksum has changed

One instance shows why re-resolution must be independent of the engine. Asked for human SOD1, the engine returned a genuine, resolvable UniProt identifier for the wrong protein, an unreviewed 134-residue record instead of the reviewed 154-residue entry; a test the engine had written for itself would have passed it. Independent re-resolution flagged it and the run was refused before the judge saw it [20]. Fluent confidence and correctness are uncorrelated, in the model [6] and in the developer [2], so the process has to supply the correlation.

7. Two example applications

The two systems are independent applications within the scope of Section 2. For each: what it does, how the method and architecture are used in it, and why the conventional build, a coding agent producing a many-agent product, was not taken. Measured results are reported separately [21,24].

7.1 Compound Insights

What it does. A cited dossier for a compound, read by scientists deciding what to make of it. The design spans the evidence range: a data-rich approved kinase inhibitor, an oral GLP-1 agonist with thin structured records so inference carries more of the dossier, and an investigational agent with no ChEMBL entry, where the question is whether the engine declines where it should. That third compound is M4 in practice: the decision that most of its dossier would read no data found, and that inference from analogues would not be permitted, was made by the operator only after the first run showed what the sources held. Both

sets of requirements hold; the work is being submitted for publication, so the record of what was tried and falsified must survive (M2).

How it is built and structured. The method of Section 4 applies without exception. The engine is pinned under a standing instruction that it is a vendor service the application must not touch; the move to the current engine version was run as a recorded acceptance campaign with a verdict comparison against the previous version, and the caveat that qualifies the comparison travels with it. Values are regenerated through evaluation campaigns. The application calls the engine through one measured path, and every value carries its provenance record on the dossier page. The earlier framework-composed design, retrieval, inference and writing as separate agents, was retired because its values could not be attributed to one call path and an error in one agent became the next agent's input.

Why not the conventional build. A coding agent would have built this faster and would have removed the upload-snapshot error class of Table 3. It was not used because the operator must be able to read and defend every dossier mechanism to a co-author and a reviewer (M1), which the interactive method guarantees by construction and an agent transcript does not, and because the specification, cold-read and register record the publication needs is produced by the method itself (M2); a harness persists progress, not the reasoning behind rulings. The many-agent product, retrieval agent, chemistry agent, writer and critic converging on a dossier, is what was tried in framework form and retired.

7.2 The SIM Framework

What it does. Decision support for the DC decision, for a nomination committee, in collaboration with Lenovo IDG Commercial Workstation Industry and Solutions [24]. For each claim in the package: the strongest case that the evidence is sufficient, the strongest case that it is not, and an adjudication, repeated R times over the same fixed evidence with the spread reported. Precedent is handled through authored negative-precedent records and a written comparability basis. It emits no go, no no-go, and no statement that a criterion is met. In the demonstration the candidate is synthetic, its readouts authored with a designed weakness, and the precedents are real, so the system is exercised on a hard case without asserting anything about a real molecule. R1 holds in its sharpest form: the output feeds a decision that commits a program to IND-enabling toxicology, and R4 is the design rule that the system emits no verdict.

How it is built and structured. The same method, with two features the decision imposes. A scope document is the single source of truth: every ruling is numbered, recorded before it takes effect, and cited by number later; where a later note disagrees with the scope, the scope wins. The engine's runs take hours per claim set, so the operator's turn rate is not the bottleneck and the interactive method costs nothing in throughput here. Every tool call, query and payload is recorded with the run. M4 applies here through precedent: that the authored negative-precedent records could not be attributed by the provenance layer, because their public references named no parseable accession, was discovered by running the demonstration, and the ruling to publish the gap as a limitation rather than change the reference format between runs was the operator's. Limitations are carried on every published surface beside the evidence they qualify, which is the misses-are-kept rule applied to the product; the committee's pages are generated from served data files with published digests, which is the checked-write discipline applied to publication. The design target is a federated deployment across workstation and edge nodes, and the demonstration runs on one machine; in the federated design the units are deployments of the same fixed roles, not autonomous agents converging on an answer.

Why not the conventional build. The natural agentic design is an orchestrator with a retrieval agent, a proponent, a critic and a judge converging on a recommendation. That output is the one the system must not produce: a committee handed a converged recommendation has been handed a verdict, the coordination failures of Section 3 would be invisible to it, and the spread it needs is what such a design removes. The parallelism it would add is unusable because the engine serializes. A coding-agent build was rejected for the reasons in 7.1, and because the scope document requires that every ruling be made by the scientist and recorded before it takes effect; an agent that proposes and executes in one step inverts that order.

8. Discussion

8.1 What the approach gives up

Three costs are real. The model does not read the live repository; it reads uploads, and the most frequent build-time failure (Table 3, row 1) follows from that; a coding agent removes the class entirely. Throughput

is bounded by the operator's turn rate, so a coding agent would expose new failure classes sooner. And the method does not transfer: its rules are prose one person enforces, where hooks could be run by anyone. These govern how fast the method finds its own defects and whether a second engineer can run it, not whether the delivered software is correct or its outputs defensible.

8.2 Limits of the evidence

This is a first-person account by a single practitioner of two systems, and the assessments in Tables 1 and 2 are made by the party they favor. There is no controlled comparison against a coding-agent build of the same system or a many-agent implementation of either task. The failure catalogue is complete for two project records and unknown beyond them. Measured results are reported separately [21,24]; the evidence here is the design rationale, the failure catalogue and the literature. The rows conceding advantage to agentic tooling make the assessments falsifiable: a reader who disagrees with a row can name the measurement that would settle it. A many-agent implementation of the same dossier task on the same compounds, with citation rows classed the same way, would move the architecture claims from assessment to evidence.

8.3 When agent frameworks are right

Nothing here argues against agent frameworks in general, and autonomous scientific agents that plan and run experiments [26,27] succeed on tasks where a wrong step is cheap to detect. The same operator's ScienceClaw platform [25] runs autonomous monitoring and synthesis workflows over oncology trial registries and news under an orchestration framework, with deterministic scripts for counting and scoring, a second model checking every outbound communication against a scope document, and a human approving each expansion of scope. There the unit of work is a surveillance finding checked before it is sent, and no committee owns it as a decision; an agent framework is the right design. The argument here is scoped by R1 to R4 and M1 to M4: when they hold, what a many-agent design gives up is what the product exists to provide.

8.4 Four questions to ask of any decision-support system

Four questions follow from R1 to R4 and apply to any AI decision-support system in this setting, whoever built it: which version of which engine produced the claim; was it replicated; where did each citation come from; and who decided. A system that cannot answer them has produced a plausible answer, which under R1 is not enough.

9. Conclusion

For decision support in pre-clinical drug discovery, agentic engineering is right about process and wrong about architecture. Its process claim is implemented here in full with one substitution: the operator executes the deterministic layer, at a cost in throughput, transfer and live-repository access that does not touch correctness or provenance. The method meets the requirements of design, development and test, and the architecture those of the output at run time; the method is a loop because the setting demands one: the science and data decisions that shape a decision-support system are forced by what the sources hold, are discovered only by building and testing against them, and are made by a scientist, one ruling at a time, recorded before it takes effect. Its architectural pattern, many role agents converging on an answer, would remove attribution and reproducibility, the two properties a system whose output a scientist or committee acts on exists to provide, in exchange for parallelism a serialized engine cannot use. One pinned engine, a provenance record on every value, replication published as spread, and a human who decides is the smallest architecture that carries the adversarial structure the decision needs, and the smallest architecture is what the evidence favors. The comparisons remain design assessments against a stated standard, and are offered as such.

References

[1] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, I. Stoica. Why do multi-agent LLM systems fail? NeurIPS 2025 Datasets and Benchmarks Track. arXiv:2503.13657.

[2] J. Becker, N. Rush, E. Barnes, D. Rein. Measuring the impact of early-2025 AI on experienced open-source developer productivity. METR, July 10, 2025. arXiv:2507.09089.

[3] J. Moreira. IACDM: Interactive Adversarial Convergence Development Methodology: a structured framework for AI-assisted software development. arXiv:2604.16399, 2026.

[4] J. C. Davis et al. Cheap code, costly judgment: a case study on governable agentic software engineering. arXiv:2607.01087, 2026.

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao. ReAct: synergizing reasoning and acting in language models. ICLR 2023. arXiv:2210.03629.

[6] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, D. Zhou. Large language models cannot self-correct reasoning yet. ICLR 2024. arXiv:2310.01798.

[7] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, I. Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. NeurIPS 2023 Datasets and Benchmarks Track. arXiv:2306.05685.

[8] A. Panickssery, S. R. Bowman, S. Feng. LLM evaluators recognize and favor their own generations. NeurIPS 2024. arXiv:2404.13076.

[9] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. ICML 2024. arXiv:2305.14325.

[10] A. P. Smit, N. Grinsztajn, P. Duckworth, T. D. Barrett, A. Pretorius. Should we be going MAD? A look at multi-agent debate strategies for LLMs. ICML 2024, PMLR 235:45883-45905. arXiv:2311.17371.

[11] J. W. Scannell, A. Blanckley, H. Boldon, B. Warrington. Diagnosing the decline in pharmaceutical R&D efficiency. Nature Reviews Drug Discovery 11, 191-200, 2012.

[12] D. Cook, D. Brown, R. Alexander, R. March, P. Morgan, G. Satterthwaite, M. N. Pangalos. Lessons learned from the fate of AstraZeneca's drug pipeline: a five-dimensional framework. Nature Reviews Drug Discovery 13, 419-431, 2014.

[13] B. Zdrazil, E. Felix, F. Hunter, et al. The ChEMBL Database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. Nucleic Acids Research 52(D1), D1180-D1192, 2024.

[14] G. K. Sandve, A. Nekrutenko, J. Taylor, E. Hovig. Ten simple rules for reproducible computational research. PLoS Computational Biology 9(10), e1003285, 2013.

[15] Anthropic. Effective harnesses for long-running agents. Anthropic Engineering, November 26, 2025. https://www.anthropic.com/engineering

[16] Anthropic. Harness design for long-running application development. Anthropic Engineering, March 24, 2026. https://www.anthropic.com/engineering

[17] Anthropic. How we built our multi-agent research system. Anthropic Engineering, June 13, 2025. https://www.anthropic.com/engineering

[18] H. He and Thinking Machines Lab. Defeating nondeterminism in LLM inference. Thinking Machines Lab: Connectionism, September 2025. doi:10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

[19] L. Moreau, P. Missier (eds.). PROV-DM: the PROV data model. W3C Recommendation, April 30, 2013. https://www.w3.org/TR/prov-dm/

[20] R. Singh. AI coding: a practical approach for scientists. Talk and companion site, BAGIM, September 17, 2026. https://bagim-ai-coding.vercel.app/

[21] R. Singh. Compound Insights: rigorously cited drug-compound dossiers. https://www.compoundinsights.dev/

[22] R. Singh. project fred: a reasoning core for continuous knowledge-graph inference in pre-clinical drug discovery. https://www.projectfred.dev/

[23] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. H. Chi, S. Narang, A. Chowdhery, D. Zhou. Self-consistency improves chain of thought reasoning in language models. ICLR 2023. arXiv:2203.11171.

[24] R. Singh. The SIM Framework: decision support for the Development Candidate gate. Overview, Science Report and Provenance pages, gate_20260917T182405Z. https://sim-ai.dev/

[25] R. Singh. ScienceClaw: autonomous life sciences research platform. https://scienceclaw.ai/

[26] D. A. Boiko, R. MacKnight, B. Kline, G. Gomes. Autonomous chemical research with large language models. Nature 624, 570-578, 2023.

[27] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, D. Ha. The AI Scientist: towards fully automated open-ended scientific discovery. arXiv:2408.06292, 2024.

[28] W. H. Walters, E. I. Wilder. Fabrication and errors in the bibliographic citations generated by ChatGPT. Scientific Reports 13, 14045, 2023.