

Decision Support for Pre-clinical Drug Discovery Built Without Coding Agents or Agent Teams: One Pinned Reasoning Engine and a Human Who Decides

Raminderpal Singh

Independent Researcher

raminderpal@hitchhikersai.org

Abstract. Two practices define AI-assisted software development in 2026: coding agents that read a repository and execute changes under automated checks, and products built as teams of role agents that a model coordinates at run time. This paper examines both for one class of system, decision support for pre-clinical drug discovery. There, a scientist or a committee acts on the output, and the reasoning engine is a language model whose outputs vary between runs. It makes two claims. On how the software is built, it adopts the process discipline the agentic engineering literature describes (verification outside the model, a person holding judgment, failures converted into standing controls) and differs in two respects. The operator runs every command on the target machine, and each science decision the data forces during the build is put to a scientist and recorded before it takes effect. On how the product is structured, it uses one reasoning engine in a topology fixed before run time, a provenance record on every value, replication published as a spread, and no go or no-go output, in place of agents that plan and route at run time and converge on an answer. The paper states requirements for the run-time output and for the build method, describes how two independent applications, Compound Insights and the SIM Framework, apply the approach. It catalogs observed failures and the controls that now catch them, and states what the approach costs. The comparisons with agentic practice are the author's design assessments against stated requirements, and the paper names the measurements that would test them.

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

1. Introduction

Two practices define AI-assisted software development in 2026. The first is the coding agent: a language model with read and write access to a repository that executes commands in the developer's environment, under hooks that block rule violations, inside a harness that carries state across context windows [1,2]. The second is the agent team: an orchestrating model that plans a task at run time, dispatches it to planner, researcher, critic and writer agents, and merges what they return [3,4]. We use "agentic engineering" for the first practice and "agent team" for the second, because the two are often discussed together and are separable.

The empirical record behind both is mixed. A randomized trial of early-2025 tools found that experienced open-source developers took 19% longer with AI assistance while estimating afterwards that it had made them 20% faster [5]; the same group has since changed its study design and reports raw results with some evidence of a speedup, and cautions that selection effects may hide larger gains [6]. A taxonomy of multi-agent failures reports that performance gains of such systems on popular benchmarks are often minimal; it groups their failures into system design, inter-agent misalignment and task verification [3]. A 2026 case study and a 2026 methodology paper agree that the process, whichever tool is used, determines whether AI-assisted development can be governed: verification outside the model, a person holding judgment, and recurring failures converted into standing controls [7,8].

This paper asks what follows when the software is decision support for pre-clinical drug discovery, where a scientist or a committee acts on the output. It makes two claims and keeps them separate. The first concerns the build: the process discipline above is correct for this setting and is adopted in full, with two differences, one in where commands run and one in who makes the science decisions that surface during the build. The second concerns the product: agent teams that plan and route at run time and converge on an answer are the wrong architecture for this setting, and a single reasoning

engine in a topology fixed before run time, with provenance on every value and disagreement published, is the alternative described. Neither claim is an argument against agent frameworks in general; Section 9.3 describes a platform by the same author for which an agent framework is the right design.

The evidence is of three kinds: the design rationale, a catalog of failures observed in two project records with the controls that now catch them, and the published literature. Two independent applications illustrate the approach: Compound Insights, which builds cited dossiers on drug compounds, and the SIM Framework, which supports the Development Candidate (DC) decision. They have separate scopes, codebases and deployments, and each deploys the same reasoning engine, project fred, as a separate pinned dependency. Measured results for each application are reported separately [9,10]. Section 2 states the requirements; Section 3 summarizes what the literature on agentic engineering and agent teams establishes; Sections 4 and 5 describe the build method and the product architecture, each with a comparison table; Section 6 catalogs failures and controls; Section 7 describes how each application applies the approach; Section 8 covers related work; Section 9 states costs and limits; Section 10 concludes.

2. Setting and requirements

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 pipeline attrition link late failure to the quality of evidence and decisions at these earlier points [11], and the long-run decline in R&D efficiency to systematic causes in how projects are chosen and progressed [12]. Each decision is contested, and a person or a committee resolves it. Decision support in this setting has one job, to assemble the evidence for and against each claim with its provenance visible, and one prohibition, to leave the decision to the person or committee that owns it.

Two sets of requirements follow. The first concerns the output at run time.

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 it rests on and to the engine version and model identity that produced it.

R3, reproducible within stated limits. The engine is run by a language model whose outputs vary between runs [13]. A value is therefore reported with its engine version, its model identity and the spread across repeated runs. Replication measures stability; correctness is established by checks outside the engine.

R4, undecided. The system assembles evidence and does not make the decision. Disagreement is shown to the decision-maker and left unresolved by the system.

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 they must survive into publication.

M3, externally verified. A model's review of its own output, without external evidence, does not verify it [8,14]. Every change is therefore checked outside the model, and every prediction about the effect of a change is written down beforehand and scored afterwards.

M4, scientist-ruled. The science and data decisions that shape the system cannot all 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. A data decision (which source, which field) or a coding decision (which

fallback) can be delegated under a written 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, and the product architecture of Section 5 to meet R1 to R4. The same habits serve both: the register that meets M2 is the same practice as the provenance record that meets R2, and the replication campaign that checks a change under M3 is the instrument that measures the stability of a value under R3. In the author's assessment, agentic engineering as described in Section 3 fully meets M3 and partially meets M1, M2 and M4; an agent team meets R2 to R4 only in part, because provenance is kept per agent, the path through the team is chosen at run time, and the output is a converged answer where R4 asks for a decision left open.

The position of the paper: build what the decision needs and nothing beyond it; put a person at every point where scientific judgment is exercised and let that person decide; and fix the architecture before run time, as a core with layers built on it, rather than letting a model choose the workflow as it runs. When a model chooses its own path, the path varies from run to run. For this class of system that variation is a defect even when the answer is right, because who decided and what the answer rests on then have to be reconstructed after the fact.

3. What the literature on agentic engineering and agent teams establishes

3.1 The coding agent and its process claim

The tooling of agentic engineering 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 version-control history [1,2]. The process claim can be stated without the tooling. Moreira's methodology holds that, absent external tool use, a language model cannot determine whether what it generated is correct, so verification must sit outside the generator, at approval points the agent may request and may not pass on its own [8]. Davis et al. report from a twelve-week instrumented case that development velocity surfaces recurring classes of failure and that engineering judgment sustains the velocity by converting those classes into governance mechanisms [7]. Two further results bear on verification. Models do not reliably correct their own reasoning without external feedback [14], and models used as judges show position and self-enhancement biases and favor their own generations [15,16]. The randomized trial cited in Section 1 shows why self-report is not evidence of benefit: developers' post-study estimates of their own speedup were in the opposite direction from the measured effect [5].

3.2 Agent teams

An agent team decomposes a task across role agents that pass state to one another and converge on an answer. The MAST taxonomy, fourteen failure modes in three categories, came from expert analysis of 150 execution traces and was applied to more than 1,600 annotated traces from seven frameworks; two of its task-verification modes are no or incomplete verification and incorrect verification [3]. Multi-agent debate improves factuality and reasoning on some tasks [17], but debate strategies do not reliably outperform self-consistency or ensembling and are sensitive to their agreement settings [18]. Anthropic's account of its multi-agent research system places the benefit in parallel exploration of independent subtasks and the cost in coordination and token use [4]. Two features of agent teams matter for the argument that follows: a model decides the path through the team at run time, and the team's positions are merged into one answer.

4. The build method

4.1 Who does what

Both applications were built by one operator, a systems engineer by training and not a professional developer, in interactive sessions with Anthropic's Claude models in the Claude.ai application, alongside a terminal [19]. The division of labor is fixed. The model drafts specifications, code, documents and checks, and runs code only in its own sandbox, against copies of files the operator uploads; it has no access to the operator's machines or repository. The operator runs every command on the target machine, reads each change in its dry run before applying it, and makes every science

and governance decision. No coding agent in the sense of Section 1 was used. Where this paper says the operator executes, it means this division of labor.

Mollick distinguishes three modes of working with a model [36, pp. 61-68], extending the centaur and cyborg pair introduced in his earlier work [33,34]. Centaur work keeps a clear division of labor, with distinct boundaries between what the person does and what the model does and clean handoffs between them; cyborg work weaves the two together; conjurer work delegates a task and evaluates the result afterwards. In both of the first two the person stays in the loop throughout, and Mollick's rule for choosing is that centaur work fits when part of the task requires judgment the model does not have. On judgment this method is centaur work: the model drafts and the scientist rules, and the handoff is never blurred. On execution it is the reverse. Mollick reports Doctorow's term for the opposite arrangement, the reverse centaur, a machine that uses a person as its appendage, working at the machine's pace and handling what the machine cannot yet do on its own [36, p. 125; 35]. That is a fair description of the execution layer here: the model writes the change script and the operator types, runs and pastes it back. Doctorow and Mollick describe the arrangement as one imposed on workers; here it is chosen, because the hands are where the checks sit. The operator reads every dry run and every result, and nothing is written to the target machine that a person has not seen. The two arrangements are one division of labor seen from its two ends, and the method depends on keeping them distinct: the scientist is never the machine's hands on a science decision, and the model is never the judge of its own change. Conjurer work, where the person is absent while the model works and judges the deliverable at the end, is the mode of the agent team and of what Mollick, following Shapiro's five-level scale, calls the dark factory, in which no human writes or reviews code [36, p. 129]; on that scale the method sits at Level 4 on authorship, since the operator writes specifications and no code, and deliberately below it on execution, and the paper's argument is that for this class of system the conjurer mode gives up what the output must carry.

The division is fixed; the way it is exercised is not. The build spanned several model versions, and the application around the model, its tools, memory, project instructions and sandbox, changed between them. Each change altered what the model could be trusted to draft unaided, how much of a session's state it could carry, and where the operator's attention was best spent. Mollick's premise is that a model is sometimes, but not always, better than the person working with it, that the person's expertise is the quality filter on its output, and that trust without verification is how a person falls asleep at the wheel [36, p. 99]; the balance between the two moves with each model and application release. The method expects this: the standing instructions and the register are the parts that persist, and the share of work the model takes within the fixed division is revised as the application and the models behind it change.

Figure 1 shows the two loops of the method. The change loop governs every edit to code or documents. The ruling loop governs the decisions that the sources force during the build (M4). Both write to a register that carries state between sessions.

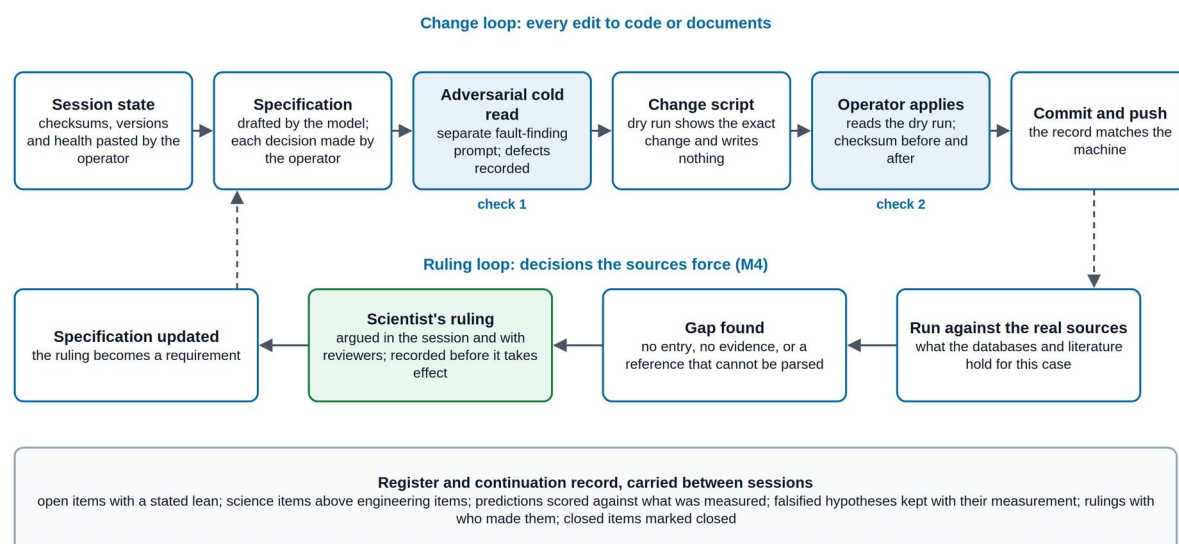


Figure 1. The build method. The change loop (top) governs every edit: a specification is accepted only after an adversarial cold read (check 1), and a change is written only after the operator reads its dry run (check 2). The ruling loop (bottom) governs the decisions the sources force: a run against the real sources exposes a gap, a scientist rules on it, and the ruling becomes a requirement before the next change. The register carries state between sessions.

4.2 The elements

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 correction goes into one of those files rather than into the next message. Harness designs make the same move with progress files [1]; here the operator 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, withdrawn claims stay in the document as a record, and only then is code written. On one Compound Insights specification, that cold read found eleven defects, five of them structural, before any code existed [19].

Checked change scripts. Every change is a script the model wrote and the operator ran. The script states the checksum it expects, shows in a dry run exactly what it would change, and aborts if the file differs from what it expects. The operator reads the dry run and applies it, checksums confirm the result, and the change is committed and pushed. One script aborted correctly when it found four matches where it expected two; the extra two were historical record [19].

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 the conversion of failure classes into governance that Davis et al. describe [7], done by hand.

The step-back. Between sessions the operator forces a review that a session does not produce on its own: an open-items register with a stated lean on each item, science items ranked above engineering items, closed items marked closed, and a continuation document. Without this review a project drifts toward whatever is most tractable; the step-back re-asserts priority.

Decisions surfaced by the data. M4 is why the method has a ruling loop. A run against the real sources shows 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 that no data were found 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's declining to answer is the correct result. A language model adds a second layer of uncertainty: its behavior on thin data (whether it fills gaps, declines or over-generalizes) has to be

measured, because it cannot be predicted in advance. 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, and run again.

A coding agent working alone resolves such gaps as it goes, choosing a fallback, and the choice is visible afterwards only if it was logged. An agent team resolves them at run time, and may resolve them differently on each run. 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 to the engineer is a prompt for approval. A science decision is not settled that way. Whether a compound with three PubMed records should carry any inferred claim, 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 in the session, 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 in the session and the register, and keeps the name of the person who made the ruling attached to it.

4.3 Comparison with coding-agent practice

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

Table 1. The build method compared with coding-agent practice (author's assessment).

Dimension	This method (interactive sessions; operator executes)	Coding-agent practice (repository access and hooks)	Assessment
Who makes science and governance decisions	The operator, one ruling at a time, recorded before it takes effect	The engineer at review points; the agent proposes. In agent teams, judgment is spread across role prompts	Equal with a single agent; this method compared with agent teams
Science and data decisions that surface during the build (M4)	Surfaced by a run against the sources, ruled on by the operator, recorded, and written into the specification	Resolved by the agent as it works unless a hook stops it; visible afterwards if logged	This method
Verification that a change is correct	Checksum, anchor count, self-tests and scratch render; the operator approves the dry run	The same checks, expressed as hooks that block the action	Equal
Where commands run	The model runs code in its own sandbox on uploaded copies; the operator runs every command on the target machine	The agent runs commands in the developer's environment; the engineer approves	Coding agents on speed; this method on the operator's knowledge of what ran
Model reads the live repository	No; it reads uploads, the source of the most frequent build-time error (Table 3)	Yes	Coding agents
Record of predictions, misses and falsified hypotheses	Kept in the register with the measurement	Kept where the harness persists it [1,2]; otherwise lost at context compaction	Equal where the harness persists it
Throughput	Bounded by the operator's turn rate	Higher on execution	Coding agents
Operator's understanding of the system	Every change is read in its dry run before it is applied	Depends on how much of the agent's work the engineer reads	This method
Transfer to a second engineer	Not possible today: the rules are prose that one person enforces	Possible where the rules are hooks	Coding agents

On who decides, how a change is verified and what is recorded, the method and a well-configured coding agent implement the same process. They differ in where commands run, which costs the method speed and transfer, and in the handling of the science decisions the data forces during the build, which the method puts to a scientist as each one arises.

5. The product architecture

5.1 *One reasoning engine, and what pinning covers*

Each application calls one reasoning engine, project fred, at version 0.3.0 [20]. fred is a ReAct-style loop [21]: the model reasons about what to check next, calls one of ten registered tools (lookups against UniProt, PubChem, PubMed, Open Targets and Reactome, a web search, and utility tools), reads the result, and repeats until it can answer or its step budget is spent. The budget defaults to 30 steps and the caller can set it between 1 and 200. fred serves one question at a time.

For every tool call fred records the tool, its arguments, the time, the duration, whether it failed, a 500-character excerpt of the reply, and the byte length and SHA-256 hash of the complete reply. The complete reply itself is not retained; recovering it means asking the database the same question again, and the hash shows whether the answer has changed. A value recalled by a model can be invented. A recorded lookup can be checked by a reviewer who trusts neither the operator nor the model.

fred is deployed as a sealed, versioned container, separately for each application, and each application treats it as a vendor service: it calls fred through one path and never edits it. An earlier Compound Insights design used a general agent framework; it was retired in favor of the single engine with one call path, so that no framework-level silent failure remains.

Pinning fred fixes its code, its tool registry and its wire protocol. It does not fix the language model. fred v0.3.0 has one model backend configured, Kimi K2.6, served from the cloud through a local Ollama daemon; fred sets no sampling parameters, and its run record states that temperature, top-p and seed were left unset [20]. The run record carries the model provider, the model name, and the tokenizer name and revision, so the model identity is recorded on every run. It is not controlled: the weights and serving stack of a hosted model belong to its provider. Outputs therefore vary for three reasons: sampling at the provider's default settings, variation that serving introduces even when sampling is greedy [13], and changes the provider makes to the model. R2 is met by recording the model identity. R3 is met only in the limited form stated in Section 2: recorded identity plus a measured spread. A deployment with a locally served model at fixed weights and fixed sampling settings would narrow the second and third sources of variation, and that deployment has not been built.

Figure 2 contrasts this arrangement with an agent team.

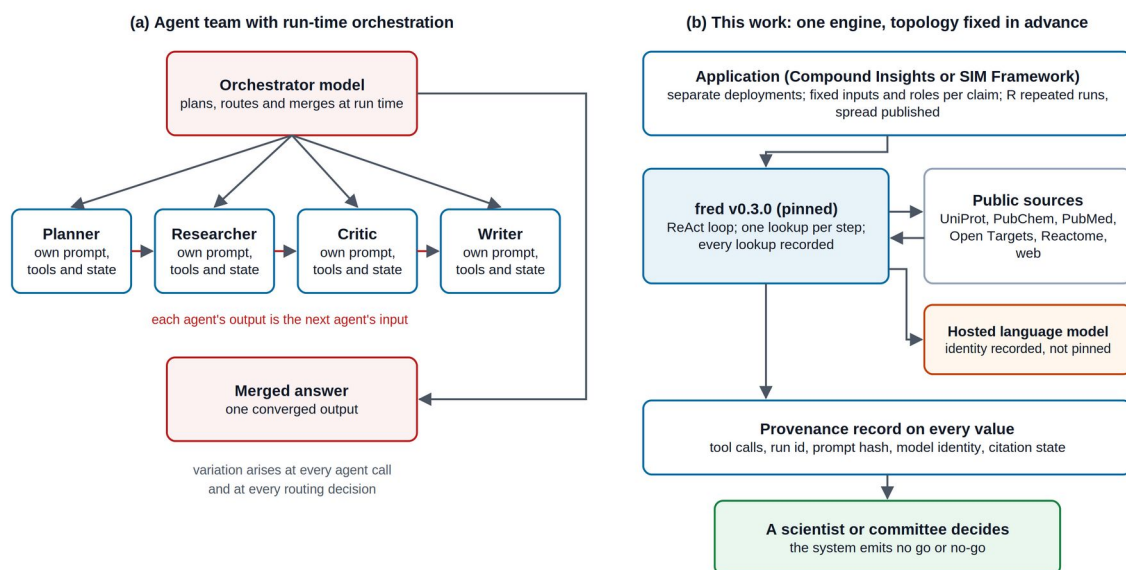


Figure 2. Product architecture. (a) An agent team with run-time orchestration: an orchestrating model plans and routes work among role agents that pass state to one another, and merges their outputs into one answer; variation arises at every agent call and every routing decision. (b) This work, deployed separately in each application: fixed inputs and fixed roles per claim, one pinned engine whose every lookup is recorded, a hosted model whose identity is recorded but not pinned, replication published as a spread, a provenance record on every value, and a person who decides.

5.2 Provenance on every value

Every inferred value in a Compound Insights dossier carries the prompt version, the run identifier, the record of lookups, the validation state of each citation on independent re-resolution, and the result of a grounding check. 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 that appears in no supplied record and no recorded search [10]. The third class matters most for safety, and it is published. Language models are documented to produce fabricated and erroneous citations [22], which is why every citation is re-resolved and, in SIM, classed. Resolution establishes that a record exists; it does not establish that the record supports the proposition it was cited for, and the SIM Framework's published record says so [10]. 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". This applies provenance practice from computational science [23,24,25] value by value.

5.3 Replication and checks outside the engine

Replication measures stability. It does not measure correctness, because a consistent error replicates. The SIM Framework's published record makes the point directly: agreement across repetitions that share one model, one prompt and one evidence base is reproducibility and should not be read as corroboration [10]. Correctness has to come from checks outside the engine.

fred's release suite is one such set of checks, run by a separate harness against the deployed service [20]. Each test question has an answer that is a small set of exact values a database holds, queried directly and hashed before the question is put to fred. Every identifier in every answer is classified by where it came from and re-queried against its database. A judge from a different model family (GLM-5.3-Flash) scores each answer after being calibrated on deliberately corrupted answers. A comparison against a stored baseline refuses to run if the case definitions, the judge prompt, the judge calibration or the harness versions have changed, so a run cannot be declared equivalent after the yardstick has moved.

Compound Insights runs evaluation campaigns of the same design against its own dossier questions. The SIM Framework argues every claim R times and publishes the spread. Both use self-consistency [26] for measurement rather than for selecting an answer: the spread is published and is not collapsed by majority vote. A change of engine version is a deliberate re-pin, recorded per claim, and outputs

produced under an earlier version are never back-filled. Compound Insights moved from fred v0.2.0 to v0.3.0 on September 19, 2026, and both applications now call v0.3.0.

5.4 The SIM Framework: three fixed roles and a committee

The SIM Framework runs the engine in three fixed roles for each claim in a nomination package (Figure 3). A constructive opening argues that the evidence for the claim is sufficient, and a skeptical one that it is not, each from the same fixed evidence menu. An arbiter reads both openings, checks their citations, and records a ruling for one position or declines to rule. The menu is the same for both openings on every repetition, and nothing ranks or selects evidence. The sequence is repeated *R* times (five in the published demonstration) and the rulings are published as a spread.

This is a multi-agent debate in form: a proponent, an opponent and a judge, the structure studied in [17,18]. It differs from the agent teams of Section 3.2 in four respects, each fixed before run time. The topology is fixed: no model decides which role runs next or what it sees. The inputs are fixed: both openings argue from the same menu. The positions are not merged: each repetition's ruling is tallied and the spread is published, so a minority position stays visible, and a repetition with no ruling is counted. The output is not a decision: the system never emits go, no-go, or "criterion met". In this paper "ruling" means the arbiter's output in one repetition, and "decision" means what the committee does.

The arbiter depends on the openings, so an error in an opening is an input to the arbiter; this is the handoff that MAST classifies as a source of inter-agent failure [3]. The controls are that citations are checked at the arbiter stage, a missing ruling is counted, and each opening's own failures are recorded per repetition. The published record also shows a limit of these controls: in some repetitions the arbiter cites identifiers that appear in neither opening, and whether those came from the engine's own lookups or from the model's weights is not resolved in the record [10].

Precedent is handled explicitly. A prior compound's outcome may be cited only with a written comparability basis, and negative-precedent records state what the literature does not contain, because a corpus of papers cannot carry the proposition that no paper exists. The DC committee reads the spread and the provenance and decides.

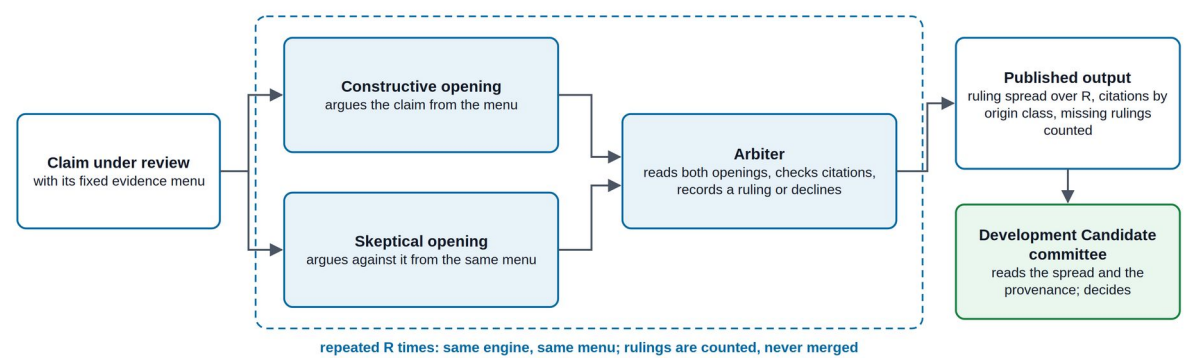


Figure 3. The SIM Framework per-claim pipeline. Two openings argue for and against the claim from the same fixed evidence menu; the arbiter reads both, checks citations and records a ruling or declines. The sequence runs *R* times on the same engine and menu; rulings are counted and published as a spread, and the committee decides.

5.5 Comparison with agent teams

Table 2 compares the architecture with an agent team, on the same basis as Table 1.

Table 2. The product architecture compared with an agent team (author's assessment).

Dimension	This architecture (one engine, topology fixed before run time)	Agent team (orchestrator plus role agents)	Assessment
Who chooses the path	Fixed before run time; in SIM, three roles per claim in a fixed order; no model routes work	An orchestrating model plans and routes work at run time	The design choice on which the other rows depend

Dimension	This architecture (one engine, topology fixed before run time)	Agent team (orchestrator plus role agents)	Assessment
Where variation between runs arises	The engine call in each fixed role, measured by replication	Each agent call and each routing decision	This architecture: fewer sources, each measured
Attribution of a value to evidence	Tool-call record, run id, prompt hash, model identity and citation state on every value; in SIM each citation classed by origin	Per-agent logs; the chain across agents is reconstructed afterwards	This architecture
Reproducibility	Engine version pinned; hosted model identity recorded but not controlled; spread published; re-pins recorded per claim	Depends on logging each agent's configuration and each routing decision	This architecture, within the limit set by the hosted model
Output to the decision-maker	Evidence and per-repetition rulings as a spread; no go or no-go	A converged answer	This architecture, under R4
Disagreement	Published as a spread; a missing ruling is counted	Resolved inside the loop; debate protocols that merge positions are sensitive to their agreement settings [18]	This architecture
Verification failures	Checks outside the engine can stop a run (Table 4)	No or incomplete verification and incorrect verification are among MAST's failure modes [3]	This architecture, where the checks exist
Throughput per claim	Serial in current deployments: one question at a time per engine instance	Parallel fan-out over independent subtasks [4]	Agent team
Breadth of task	Narrow by design: claim-level inference from literature and databases	Research, synthesis, writing and tool use in one pipeline	Agent team

The two rows that favor agent teams, throughput and breadth, are real. The throughput row depends on the current deployment: fred serves one question at a time and its model is served remotely, so there is nothing for fan-out to use. That is a property of the deployment. The R repetitions are independent and could run in parallel on several engine instances, which the SIM Framework's federated design target anticipates (Section 7.2). The case against an agent team therefore rests on attribution and on keeping disagreement visible, which are what R2 and R4 require. The task is narrow by design, because every additional step that a model chooses at run time is a step that has to be attributed afterwards.

6. Observed failures and the controls that catch them

Tables 3 and 4 list failures observed in the two project records and in fred's release testing, the mechanism behind each, and the control that now catches it. Each run-time row names the system, since a control in one system is not a control in the other. Most controls were added after the failure was observed and scored, as Section 4 describes. In MAST's terms [3], the build-time failures are specification and verification failures of a single model. The run-time failures are verification failures made visible by a check that can stop a run, and, in SIM, failures at the fixed handoff from the openings to the arbiter.

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

Build-time failure (observed)	Mechanism	Control that catches it
The model reasons from an earlier copy of a file rather than the live one	Uploads are snapshots; the live file changes; the model edits the copy it has	Checksum before and after every write; the script aborts on a mismatch
The model reports a review that was only a re-read	Checking while writing is not a review; self-correction without external feedback is unreliable [14]	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
The model announces a deliverable that does not exist	Fluent completion text does not show that the file exists	The operator looks for the file and its checksum, and disregards the sentence
The task quietly narrows or widens	Adjacent defects are more tractable than the objective, and the model drifts to them	The objective is restated at the top of every reply, with whether the reply advanced it
A change script would have touched historical record	A pattern matched more places than intended	The script asserts the expected match count and aborts otherwise

Table 4. Run-time failures observed in the two systems and in fred's release testing, and their controls.

Run-time failure (observed)	Mechanism	Control that catches it
Stalled run (Compound Insights)	An engine job neither completes nor fails; fred serves one question at a time, so a stuck question blocks the next	A poll limit; the run is recorded as failed, under the rule that nothing fails silently
Contamination on answer shape (Compound Insights)	A judge or downstream step keys on format rather than content	A judge calibrated on deliberately corrupted answers; cell-by-cell comparison with a frozen baseline
Genuine citation, wrong entry (fred release suite)	A resolvable identifier for a different entry from the one asked for (described below)	Required citations checked against a case file fixed in advance; every identifier re-queried outside the engine; the run refused before judging
Precedent used as unearned support (SIM)	A prior compound's outcome cited without a stated comparability basis	Negative-precedent records and a written comparability basis for each precedent
Precedent records the provenance layer cannot attribute (SIM)	Their public references name no parseable accession	The gap is published as a limitation beside the evidence it qualifies
Arbitration returns no ruling (SIM)	The arbiter stage completes without a ruling although both openings completed	The claim's no-ruling counter records the loss; the gap is published as a limitation
Baseline drift (Compound Insights)	The yardstick changes between campaigns	The comparison refuses to run if the stored baseline's checksum has changed

One instance shows why re-resolution must be independent of the engine. Asked for human SOD1, fred cited a genuine, resolvable UniProt accession for an unreviewed 134-residue entry instead of the reviewed 154-residue entry. The database had ranked entries differently when fred asked for one result rather than five, and fred's lookup tool does not yet prefer reviewed entries. The release suite checks that each citation its case file requires is present; the reviewed accession was missing, so the citation stage recorded a violation and refused the run before the judge saw it [20]. A check that asks only whether a cited identifier resolves would have passed this answer, because the unreviewed accession resolves. Fluency and correctness come apart in the model [14] and in the developer's judgment of the model [5], so the process has to supply the check.

7. Two example applications

The two applications are independent systems within the scope of Section 2. For each, this section states what it does, how the method and the architecture are applied in it, and why the conventional build, a coding agent producing an agent team, was not used. Measured results are reported separately [9,10].

7.1 Compound Insights

What it does. Compound Insights produces a cited dossier for a compound from public sources, including ChEMBL [27], UniProt, PubMed, ClinicalTrials.gov, Open Targets and Reactome, together with literature-grounded inference, and scientists read it to decide what to make of the

compound [9]. Its test compounds span the range of available evidence: a data-rich approved kinase inhibitor; an oral GLP-1 receptor agonist whose structured records are thin, so inference carries more of the dossier; and an investigational agent with no ChEMBL entry, where the question is whether the engine declines to answer where it should. The third compound is M4 in practice. The ruling that most of its dossier would read "no data found", and that inference from analogues would not be permitted, was made by the operator after the first run showed what the sources held. Both sets of requirements hold, and the work is being prepared for journal submission, 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. fred is pinned under a standing instruction that it is a vendor service the application must not change. The move to fred v0.3.0 was run as a recorded acceptance campaign with a comparison against the previous version, and the caveat that qualifies the comparison is kept with it. Values are regenerated through evaluation campaigns. The application calls fred through one path, and every value carries its provenance record on the dossier page. The earlier design, in which retrieval, inference and writing ran as separate agents in a general framework, 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), and because the record that publication needs, in the specification, cold reads and register, is produced by the method itself (M2); a harness persists progress, and the reasoning behind a science ruling has to be written down by the person who made it. An agent team, with retrieval, chemistry, writing and critic agents converging on a dossier, is close to the framework design that was tried and retired.

7.2 The SIM Framework

What it does. The SIM Framework is decision support for the DC decision, built for a nomination committee, in collaboration with Lenovo IDG Commercial Workstation Industry and Solutions [10]. For each claim in a nomination package it produces the strongest case that the evidence is sufficient, the strongest case that it is not, and a ruling, 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 or no-go, and no statement that a criterion is met. In the published 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. Because every readout was authored with a weakness, the demonstration has no positive control, and the published record says so [10]. R1 holds in its sharpest form, since the output feeds a decision that commits a program to IND-enabling toxicology, and R4 is the design rule that the system emits no decision.

How it is built and structured. The method is the same, with two features the decision imposes. A scope document is the single source of truth: every ruling is numbered, entered before it takes effect and cited by number later, and where a later note disagrees with the scope, the scope governs. Every tool call, query and payload is recorded with the run. M4 applies here through precedent. Running the demonstration showed that the provenance layer could not attribute the authored negative-precedent records, because their public references named no parseable accession; 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 rule that misses are kept, 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. A single claim set takes hours to run, so in SIM the engine's run time, rather than the operator's turn rate, bounds how often an evaluation campaign can be run; the build-time cost of Section 9.1 still applies. 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 further deployments of the same fixed roles, not agents that plan their own work.

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 recommendation is the output the system must not produce: a committee handed a converged recommendation has been handed a decision, the coordination failures of Section 3.2 would be invisible to it, and the spread it needs is what such a design removes. A coding-agent build was not used for the reasons in Section 7.1, and because the scope document requires every ruling to be made by the scientist and recorded before it takes effect; an agent that proposes and executes in one step reverses that order.

8. Related work

AI-assisted software engineering. The process claims adopted in Section 4 come from a methodology for AI-assisted development [8], an instrumented case study of governable agentic engineering [7], and published harness designs for long-running coding agents [1,2]. The randomized trial and its update [5,6] are the main controlled evidence on developer productivity with AI tools; neither tested the interactive method described here.

Multi-agent systems and debate. MAST [3] is a systematic account of how multi-agent systems fail. Work on debate [17,18] and on self-consistency [26] bears on how SIM repeats and adjudicates claims, and work on models as judges [15,16] bears on the choice of a judge from a different model family in fred's release suite.

Provenance and reproducibility. The provenance record in Section 5.2 applies established practice: the W3C PROV data model [25], surveys of provenance for computational tasks [23], and rules for reproducible computational research [24]. The emphasis in this work is granularity: provenance is attached to each value a model produces, including each citation, rather than to a workflow run as a whole. Work on nondeterminism in LLM inference [13] bears on what a pinned engine can and cannot promise when its model is hosted.

Language models in science. Autonomous agents that plan and run chemical experiments [28] or carry out open-ended machine-learning research [29] have been demonstrated on tasks where a wrong step is cheap to detect. The documented rate of fabricated and erroneous citations from language models [22] is the reason for independent citation re-resolution.

People using decision aids. R4 withholds the decision from the system, but withholding a verdict does not by itself protect a committee from over-reliance. Automation bias produces errors of omission and commission when decision aids are imperfect, occurs in experts and in teams, and is not prevented by training or instructions [30]; in clinical decision support it is mediated by workload, task complexity and time pressure [31]. Whether presenting a spread of rulings with provenance reduces over-reliance, or adds anchoring on the modal ruling, is not tested in this work (Section 9.2).

This paper does not survey language-model systems for drug discovery; its comparison is with the engineering practices of Section 3.

9. Discussion

9.1 *What the approach gives up*

The costs are real. The model does not read the live repository; it reads uploads, and the most frequent build-time failure (Table 3, first row) follows from that, a class a coding agent removes entirely. Throughput is bounded by the operator's turn rate, so a coding agent would surface new failure classes sooner. The method does not transfer: its rules are prose that one person enforces, where hooks could be run by anyone. These costs govern how quickly the method finds its own defects and whether a second engineer could run it. They are also not fixed: as the models and the application change, the share of work the model takes within the division of Section 4.1 changes with them, and the costs stated here are those observed over the versions the build spanned. In the author's assessment they do not reduce the correctness of the delivered software or the defensibility of its outputs; these depend on the checks, and the checks are the same in both practices. On the product side, a hosted model limits what the pinned engine can promise (Section 5.1), and the serial engine limits throughput (Section 5.5).

9.2 Limits of the evidence

This is a first-person account by one 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 either system or an agent-team implementation of either task. The failure catalog is complete for the two project records and unknown beyond them. The build-method figures in Section 4 come from the author's project records, summarized in [19], and cannot be audited independently of this paper. The identity of the model that drafted each artifact was not part of the build record described here, and applying M2 to the build would add it.

On the product side, the language model is hosted and not pinned (Section 5.1). The SIM demonstration has no positive control, and the origin of some identifiers the arbiter cites is unresolved (Sections 5.4 and 7.2). The premise that a committee is better served by a spread of rulings with provenance than by a converged recommendation is argued from R4 and has not been tested with committees [30,31].

The rows of Tables 1 and 2 that concede advantage to agentic practice make the assessments testable: a reader who disagrees with a row can name the measurement that would settle it. Two measurements would move the architecture claims from assessment to evidence. The first is an agent-team implementation of the same dossier task on the same compounds, with its citations classed the same way. The second is a comparison of committee decisions made from a spread of rulings with decisions made from a converged recommendation.

9.3 When agent frameworks are right

Nothing here argues against agent frameworks in general (Section 8 notes where autonomous agents have been demonstrated). The same author's ScienceClaw platform [32] 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 person 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 for it. The argument of this paper is limited to systems for which R1 to R4 and M1 to M4 hold.

9.4 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, and which model, produced the claim? Was it replicated, and what was the spread? Where did each citation come from, and was it checked outside the engine? Who decided? A system that cannot answer them has produced a plausible answer, and under R1 a plausible answer is not enough.

10. Conclusion

For decision support in pre-clinical drug discovery, this paper keeps the process discipline of agentic engineering and declines its two default choices. It keeps verification outside the model, a person holding judgment, and failures converted into standing controls. It declines the coding agent, at a cost in throughput, transfer and live-repository access, so that every command on the target machine is run by the operator and every science decision the data forces is put to a scientist and recorded before it takes effect. It declines the agent team, because a team that plans its own path and converges on an answer removes the attribution and the visible disagreement that a committee acting on the output needs. The alternative is one pinned reasoning engine in a topology fixed before run time, a provenance record on every value, replication published as a spread, and a person who decides. Its limits are stated: the model behind the engine is hosted, and its identity is recorded but not pinned; replication measures stability and does not establish correctness; and the comparisons are the author's design assessments against stated requirements. The measurements that would test those assessments are named in Section 9.2.

Acknowledgments

The author thanks Eric Ma for his review of an earlier draft; his corrections and comments are reflected throughout this version.

References

- [1] Anthropic. Effective harnesses for long-running agents. Anthropic Engineering, November 26, 2025. <https://www.anthropic.com/research/effective-harnesses-for-long-running-agents>
- [2] Anthropic. Harness design for long-running application development. Anthropic Engineering, March 24, 2026. <https://www.anthropic.com/research/harness-design-long-running-apps>
- [3] 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 (v3, October 2025).
- [4] Anthropic. How we built our multi-agent research system. Anthropic Engineering, June 13, 2025. <https://www.anthropic.com/research/multi-agent-research-system>
- [5] 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.
- [6] METR. We are changing our developer productivity experiment design. METR blog, February 24, 2026. <https://metr.org/blog/2026-02-24-uplift-update/>
- [7] J. C. Davis et al. Cheap code, costly judgment: a case study on governable agentic software engineering. arXiv:2607.01087, 2026.
- [8] J. Moreira. IACDM: Interactive Adversarial Convergence Development Methodology: a structured framework for AI-assisted software development. arXiv:2604.16399, 2026.
- [9] R. Singh. Compound Insights: rigorously cited drug-compound dossiers. <https://www.compoundinsights.dev/>
- [10] R. Singh. The SIM Framework: decision support for the Development Candidate gate. Overview, Science Report and Provenance pages, gate_20260917T182405Z. <https://sim-ai.dev/>
- [11] 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.
- [12] 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.
- [13] 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/>
- [14] 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.
- [15] 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.
- [16] A. Panickssery, S. R. Bowman, S. Feng. LLM evaluators recognize and favor their own generations. NeurIPS 2024. arXiv:2404.13076.
- [17] 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.
- [18] 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.
- [19] R. Singh. AI coding: a practical approach for scientists. Talk and companion site, BAGIM, September 17, 2026. <https://bagim-ai-coding.vercel.app/>
- [20] R. Singh. fred v0.3.0: a research service that looks things up, with a record of every lookup. Release notes dated September 8, 2026. <https://www.projectfred.dev/>
- [21] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. Narasimhan, Y. Cao. ReAct: synergizing reasoning and acting in language models. ICLR 2023. arXiv:2210.03629.
- [22] W. H. Walters, E. I. Wilder. Fabrication and errors in the bibliographic citations generated by ChatGPT. Scientific Reports 13, 14045, 2023.

- [23] J. Freire, D. Koop, E. Santos, C. T. Silva. Provenance for computational tasks: a survey. *Computing in Science and Engineering* 10(3), 11-21, 2008.
- [24] G. K. Sandve, A. Nekrutenko, J. Taylor, E. Hovig. Ten simple rules for reproducible computational research. *PLoS Computational Biology* 9(10), e1003285, 2013.
- [25] L. Moreau, P. Missier (eds.). PROV-DM: the PROV data model. W3C Recommendation, April 30, 2013. <https://www.w3.org/TR/prov-dm/>
- [26] 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.
- [27] 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.
- [28] D. A. Boiko, R. MacKnight, B. Kline, G. Gomes. Autonomous chemical research with large language models. *Nature* 624, 570-578, 2023.
- [29] 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.
- [30] R. Parasuraman, D. H. Manzey. Complacency and bias in human use of automation: an attentional integration. *Human Factors* 52(3), 381-410, 2010. doi:10.1177/0018720810376055.
- [31] K. Goddard, A. Roudsari, J. C. Wyatt. Automation bias: a systematic review of frequency, effect mediators, and mitigators. *Journal of the American Medical Informatics Association* 19(1), 121-127, 2012. doi:10.1136/amiajnl-2011-000089.
- [32] R. Singh. ScienceClaw: autonomous life sciences research platform. <https://scienceclaw.ai/>
- [33] E. Mollick. Co-Intelligence: Living and Working with AI. Portfolio/Penguin, 2024.
- [34] F. Dell'Acqua, E. McFowland III, E. Mollick, H. Lifshitz-Assaf, K. Kellogg, S. Rajendran, L. Kraye, F. Candelon, K. R. Lakhani. Navigating the jagged technological frontier: field experimental evidence of the effects of AI on knowledge worker productivity and quality. *Harvard Business School Working Paper* 24-013, 2023. doi:10.2139/ssrn.4573321.
- [35] C. Doctorow. Reverse centaurs. *Locus*, September 2025. See also C. Doctorow, The reverse-centaur's guide to criticizing AI, *Pluralistic*, December 5, 2025.
- [36] E. Mollick. Co-Existence: The Next Phase of AI. Portfolio, October 20, 2026.