The Hypothesis Graph: A Verifiable Semantic Memory for Coding Agents

June Kim
Independent Researcher
june@june.kim
ORCID 0009-0005-3153-9396

May 28, 2026

Abstract

The hypothesis graph is a novel data structure for coding agents that can deepen their reasoning and make it accountable. Implemented at the harness layer, its nodes are testable claims, its edges the conditions that refute them. It updates by inquiry, whose novel mode is abduction, beyond deduction and induction. Here, we demonstrate the mechanism on one contamination-free bugfix that an agent on a minimal prompt could not generalize; we supply the missing comparator that a context-bound agent could not abduct. This enables verifiable accountability, coordinated concurrency and efficient reasoning retention at no additional training cost, employable by any coding agent harness today.

1 Introduction

In coding agents, the LLM is wrapped in a harness: the verification, testing, and memory a software task needs. The field building these is moving up a level of abstraction, from the model to the harness. Roychoudhury et al. (2025) reframe the goal as programming with trust, arguing that deployment turns on verification, testing, and analysis built into the agent rather than on raw generation (arXiv:2502.13767). Liu et al. (2024) survey agents for software engineering and organize the field around those same missing pieces (arXiv:2409.02977); Yehudai et al. (2025) add that scoring final outputs misses the reasoning and failure causes inside a run, and call for trajectory-level assessment (arXiv:2503.16416); Wang et al. (2025) name the lack of persistent, structured memory as a core limit, leaving agents to repeat errors and forget past successes (arXiv:2508.11126).

A patch passes the visible tests, but that's not enough: passing certifies only the cases the tests cover. An over-narrow patch passes them and is wrong off-suite. Confirming it means reconstructing the reasoning the agent never recorded, at a cost approaching that of producing it, so the work shifts from writing to checking. Code review is the bottleneck.

When reasoning is discarded, each run rebuilds context from scratch. Even where agent memory adopts the cognitive-architecture lineage, as CoALA (Sumers et al. 2024) does in mapping it onto Soar (Laird 1987) and ACT-R, the semantic slot stores facts rather than a falsifiable structure, so the search is discarded once a patch passes. At best, a trail of blobs is saved as provenance.

We were promised a junior developer with near-infinite patience. All we got was tool calls in a loop: a cracked-up amnesiac contractor, leaving mistakes for maintainers to review.

Problem	Solution
Reasoning cost	A domain-aware division of intellect: the model reasons, the harness holds the open hypotheses and runs the gate.
Agent trust	A verifiable reasoning chain: a claim clears only when an independent party reruns its recorded trial.

Problem	Solution
Lost reasoning	A human-legible representation: typed nodes a later run or a human reads and replays.
Review bottleneck	Every step grounded in a trial: a reviewer accepts the work or resumes where it stuck.

Here we give the agent a trail of verifiable knowledge. The hypothesis graph is a semantic memory that holds an agent's reasoning while it works. The fix arrives with the inquiry that produced it: what was hypothesized, what was tested, what was ruled out. Each step carries a trial a stranger can rerun. The promise reverses the trust assumption by shifting the burden of proof outside the agent. The model still reasons, but each node it produces is checkable on its own at the harness layer, independent of the model.

What that buys is not incremental. On one contamination-free bug, an externalized comparator carried a weaker model to a fix the strongest released model could not reach without it (§7): a two-tier capability lift, where a scaffold usually buys only automation.

2 The hypothesis graph

2.1 Requirements

A semantic memory that holds an agent's live reasoning must clear four requirements at once:

- Holds hypotheses. Stores a hypothesis still under consideration, where prose, skill libraries, and retrieval keep only verified facts and established chunks: the hypothesis-shaped gap none of them fills.
- Refutable by test. Each claim carries a kill condition: the executable test that can prove it wrong.
- Independently verifiable. A stranger verifies a conclusion by rerunning its recorded trial, instead of re-deriving the reasoning or taking the author's word.
- Persistent memory. The trail persists past the context window rather than being discarded once a patch passes.

Prior structures fill some of these roles (✓ native, ◐ with common extensions, ✗ needs another layer doing the work).

Structure	Holds hypotheses	Includes tests	Independently verifiable	Persistent memory
Hypothesis graph (this work)	✓	✓	✓	✓
Truth-maintenance (Doyle 1979; de Kleer 1986)	✓	◐	◐	◐
Provenance / lineage (W3C PROV, Moreau et al. 2013)	◐	◐	◐	✓
Search + proof tree (Clarke et al. 2000; Solar-Lezama et al. 2006)	✓	◐	◐	◐
Argumentation (Dung 1995; Modgil & Prakken 2014)	✓	◐	◐	◐
Event-sourced log / ReAct trace (Yao et al. 2023)	✓	◐	◐	✓

The middle two columns are verification (a falsifiable trial a stranger can rerun), the outer two are the hypothesis-shaped gap and retention. Only the hypothesis graph clears all four in one append-only file.

An LLM is what finally fills it. Filling the graph takes a reasoner that reads a surprising failure, proposes candidate causes in open vocabulary, and turns each into an executable test, with no hand-built domain model. Classical inference engines could do this only inside a formalism encoded by hand, which is why the slot stayed a research program; the LLM populates it across arbitrary codebases, which is what makes the structure practical here.

2.2 Graph semantics

The structure meets all four with three pieces, a node, an edge, and one invariant. A node is a claim bound to a trial: a hypothesis, the perturbation that tests it stated as an exact command, the observed outcome,

and a credence typed by the reasoning mode that established it. An edge is generated by a kill condition: the manner of a hypothesis's death names the next hypothesis, so the structure is self-extending, with no external controller deciding where to look. The invariant comes before the operations, because every operation is defined to preserve it:

> Replay invariant. Every node is reconstructible, by a stranger who does not trust the author, from its recorded trial alone: the exact command, the observed outcome, the verdict, and the credence cap.

The contract holds on the node's mechanical skeleton; the hypothesis prose the node also carries falls outside it, by design. The prose is the part an auditor would otherwise have to trust, and the recorded trial is what replaces trusting it, so the invariant draws its line exactly where checkability begins. The guarantee is narrow and named: the command, outcome, and verdict are checkable, while the mode label that caps a credence is a convention the writer is trusted to apply honestly.

2.3 Operations

Five operations maintain the structure, each defined with the one-clause argument that it preserves the invariant, the way a balanced tree's insert is defined to restore balance:

- Create (a smart constructor), append a node: abduction writes a hypothesis with its kill condition and the exact trial that tests it ("the bulb is dead", trial `swap in a fresh bulb`). Preservation: a node is admitted only if its trial replays, so by construction every committed node satisfies the contract, and the invariant holds by induction over reachable graphs. This admission rule is what makes every later operation safe.
- Read / replay: ask which hypotheses are still open, and reconstruct any node by re-running its recorded trial, no trust in the author required. Preservation is vacuous, read mutates nothing, but replay is the one that does the work, the check every other operation's preservation is stated against.
- Classify (the update): a trial's outcome marks its node killed or witnessed and caps its credence at the mode that earned it, a verdict written once. Preservation: classify appends a verdict and never edits the recorded trial, so the node still replays to the same outcome.
- Link (edge-from-kill): the manner of a hypothesis's death names the next hypothesis (the dead bulb names "the dimmer is broken", trial `bypass the dimmer to the wall`), so a classification spawns the next node with no external controller. Preservation: link only appends, and each new node is admitted under Create's rule. This is the one genuinely novel operation.
- Prune: a dead branch leaves the working frontier while its record stays in place. Preservation is trivial, prune is a frontier-set operation that changes what is live and deletes nothing, so every pruned node replays exactly as before.

2.4 Auditability

The invariant yields an essential property that no benchmark can evaluate:

> Local Replay Auditability. Any single conclusion is checkable by re-running that node's one recorded trial, without reconstructing the inquiry and without extending trust to whoever ran it.

This is, for inquiry, the analogue of a certificate a consumer checks without trusting the producer, and like the Merkle audit path or proof-carrying code, its value rests on a contract; a complexity bound is beside the point.

Two grades of it matter. Where the trial is a deterministic command over pinned inputs, a compiler on a fixed toolchain, a test in a container, replay is strong: re-execution reproduces the recorded outcome, and the lead case (§7) is here. Where the trial runs a model or a live service, replay is artifact-level: the recorded output is verified and the deterministic predicate re-run over it, the command standing as a provenance event rather than a reproducible computation. The honest scope is the first shading into the second as the system under test grows less deterministic. Pruning leaves all of it untouched: a pruned branch drops from the working frontier but stays in the record, so its nodes replay exactly as before.

2.5 Knowledge maintenance

The nodes are ordinary; what is novel is the edge semantics. A search tree finds; a proof tree justifies. The hypothesis graph is both at once, because the search path is the justification: every step was a trial.

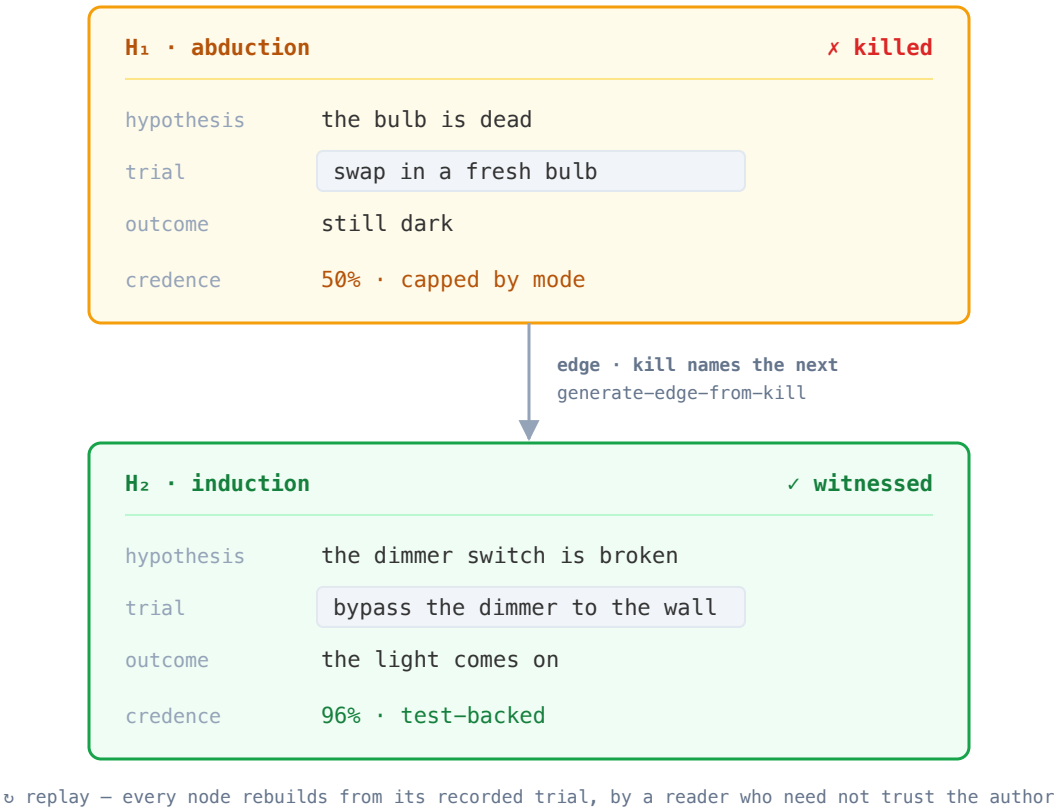

Figure 1: A hypothesis graph, two nodes and the edge between them, on the dead-light inquiry. The bulb hypothesis is killed by a cheap trial (swap in a fresh bulb, still dark); its death names the next node, the dimmer, which a second trial witnesses (bypass it to the wall, the light comes on). Each node binds a hypothesis to a trial, an observed outcome, and a credence capped by the mode that earned it: abduction proposes and stays low, induction is test-backed and rises. Every node rebuilds from its recorded trial, so an auditor replays the structure instead of trusting it.

It sits at the confluence of older lineages: truth-maintenance and model-based diagnosis (de Kleer 1986; Reiter 1987; de Kleer & Williams 1987), sequential experimental design (Wald 1947; Vovk & Wang 2021), abstract argumentation (Dung 1995), and counterexample-guided refinement (CEGAR, Clarke et al. 2000; CEGIS, Solar-Lezama et al. 2006). Refinement is the closest kin: a counterexample is a kill that names the next experiment, the hypothesis graph's defining edge. What is novel is running it over an open hypothesis space abduced in domain vocabulary. Replayability stands in for the sound abstraction a closed setting supplies for free, with completeness as the price the open move forfeits (§15).

The near neighbors each hold part of this and source the rest from outside themselves. A truth-maintenance system (Doyle 1979; de Kleer 1986) maintains belief status under assumptions, but the empirical trial and the kill-generated successor are external conventions. Provenance (W3C PROV, Moreau et al. 2013) records replayable activities, yet does not decide which hypothesis comes next. And the ReAct trace (Yao et al. 2023), the strongest mundane baseline, is an append-only log whose continuation policy the controller decides and the record never holds.

What the hypothesis graph adds is their composition in one append-only object: an open-domain hypothesis, its executable trial, its kill condition, and the successor that kill names. That object belongs to the verifiable family whose value is a contract rather than a complexity bound, certificate transparency (Laurie et al., RFC 9162), proof-carrying code (Necula 1997), content-addressed provenance: a data structure paired with the protocol that writes and checks it.

2.6 Semantic memory

This is the data structure for testable inquiry, and its entire power is the perturbation surface. Strip the ability to poke the system and read an outcome, and the same shape degrades into a plausibility tree, which is the confabulation failure mode it exists to prevent. It is also not how minds run. Minds run on simulation, fast and compressive and intuitive, but a simulation is not verifiable from outside, so inquiry that has to be checked trades it for an explicit perturbation surface. The hypothesis graph is the verifiable serialization reasoning compiles to, so it can be checked by someone who does not trust you. Proof is to intuition as the hypothesis graph is to inquiry: not the thinking, the residue of the thinking that survives a stranger's replay.

That residue is what the memory typology calls the `smem`: persistent, typed, queryable, and owned by the harness rather than the model. Those same properties make it an interface for agent interop: because the structure is typed and external, a second agent, a later run, or a human auditor reads and writes against one contract and can rerun any node rather than trust it (§8). The graph in this work is one markdown file per inquiry.

3 Actionable epistemology

Knowing is an act to update the credence of a claim. This is the subject of Verifiable Knowledge; the property that matters here, a verdict a model cannot author for itself, is the one §7 turns into a capability lift. In the Hypothesis Graph, knowledge has the following properties:

- Three states. Witnessed is true, a build presently passing; killed is false, a build gone red; open is untrue, a conjecture awaiting its test. These are the entitlement ledger's states, scoped here to one node and its kill edge.
- Credence. A node carries a credence capped by the mode that earned it, low for abduction, higher once tested (Ramsey 1926), the step a bare LLM skips when it emits uniform confidence with no propagation along the chain.
- Survived belief. A node counts as knowledge only after withstanding a trial, a verificationist criterion (Ayer 1936), indexed to the stakes of acting on it. Before trial, a claim is a hypothesis.
- Causal connections. Each node wires to what it depends on, so claims compose into a derivation, every step caused by an earlier one (Pearl 1988).

With each round of inquiry the causal boundary sharpens, and a fix that respects it generalizes past the case at hand. With each successive trial, credence accumulates to actionable confidence.

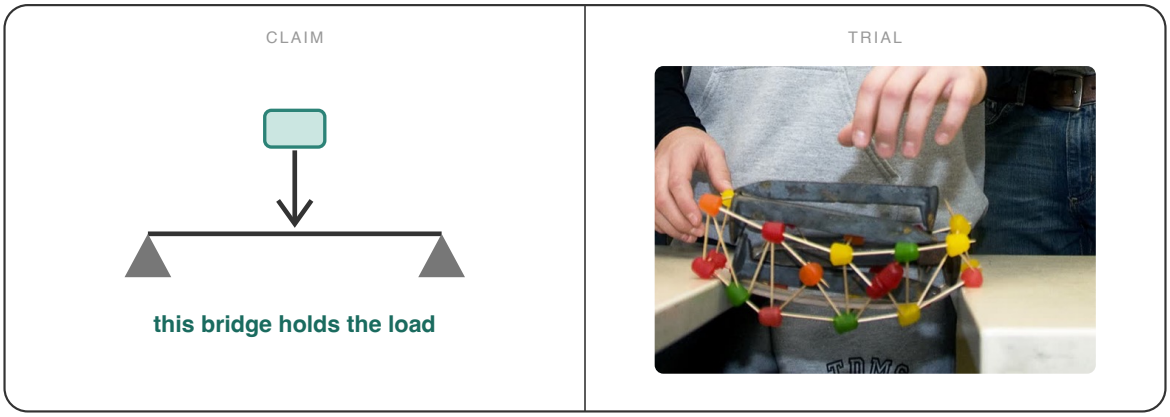

Figure 2: A claim and its trial as one record. Left, the claim: a load resting on a span, this bridge holds the load. Right, the trial: a toothpick-and-gumdrop model bridge bearing a steel weight across two supports. Photo: Oregon Department of Transportation, CC BY 2.0.

This is one more projection of the protocol Verifiable Knowledge sets out, from where the agent stands, and its payoff is transfer. Paired with its trial, a node carries its warrant across intact; handed on the author's word, only the verdict crosses and the warrant is re-derivable from scratch.

A coding agent is already a strong internal verifier, running the test and verifying that it passed. What it lacked was the protocol, extending credence beyond the context window. With the protocol, knowledge compounds for other agents and humans.

4 Methodeutics: a discipline of inquiry

A store of causal knowledge needs reasoning to be encoded into it. How can reasoning become mechanical enough for encoding? The insight, the leap to a candidate cause, stays with the model, but the harness needs a method to stage reasoning. Reasoning then becomes mechanical the way a proof is, through the discipline of checking ideas. In a precise and limited sense the harness encodes and generates reasons mechanically. This discipline, Peirce called inquiry.

His Illustrations of the Logic of Science (1878) and Pragmatism as the Logic of Abduction (1903) type the operations of inquiry into three irreducible modes.

- Abduction generates explanatory hypotheses for surprising observations: what would, if true, make this no longer surprising?
- Deduction derives testable predictions from hypotheses: if this hypothesis holds, what follows?
- Induction tests predictions against evidence: does the evidence accord with the prediction?

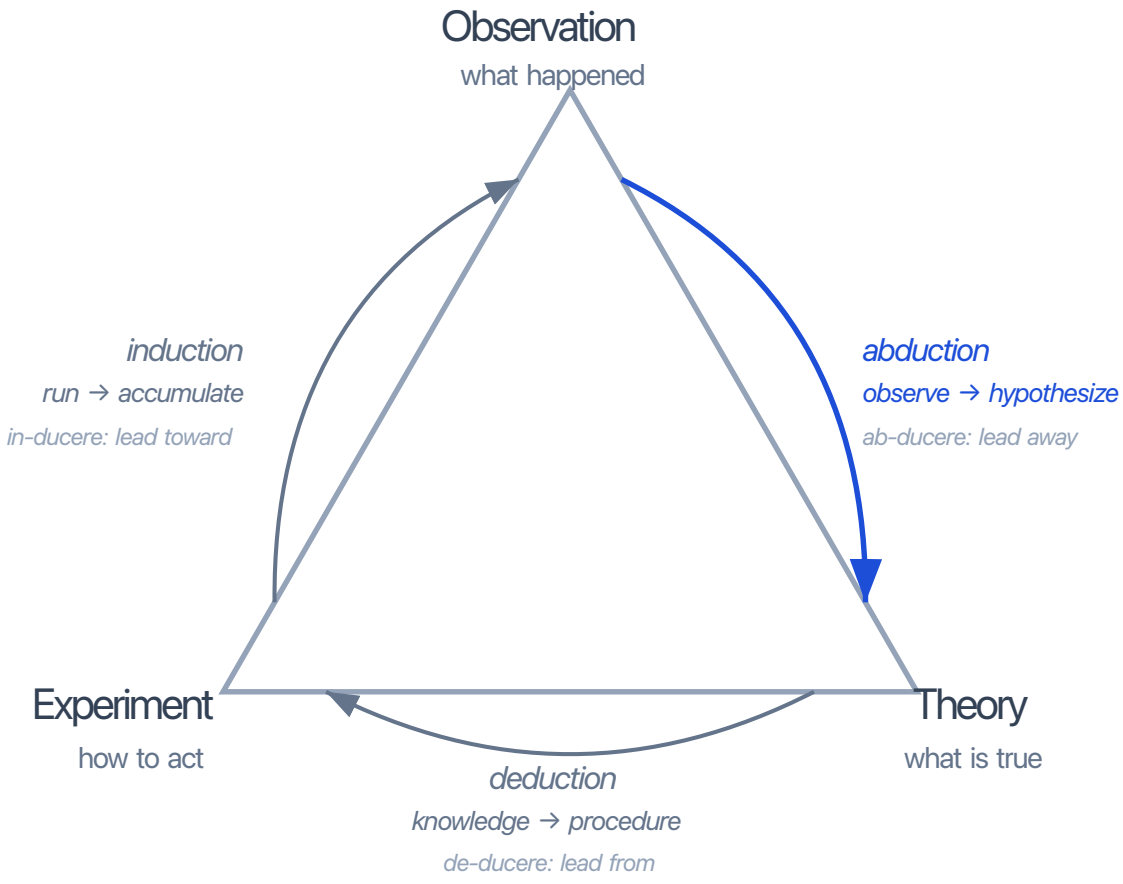

Figure 3: The three modes as one cycle: Observation → Theory (abduction), Theory → Experiment (deduction), Experiment → Observation (induction).

No single mode carries a belief to its grade. Abduction proposes content but does not test it; induction tests but introduces no new explanatory content; deduction traces consequences but invents nothing. The credence a node ends up with is what traversing all three earns it, and that is what it means to call the modes typed: each is fixed by what it can't do.

Deduction is where readers balk: why is abduction responsible for theory? Isn't that deduction's job? No. Deduction neither generates the theory nor proves it; it unfolds the hypothesis into the predictions it must answer for. The theory was abduced, the predictions deduced, and induction does the testing.

Keep them separate and each does its one job; collapse them and you get familiar failure modes:

- Confirmation bias: induction without abductive alternatives
- Confabulation: abduction without inductive grounding
- Free-association: no typed mode at all

That collapse is exactly what modern LLM agents do by default, since a single forward pass proposes, predicts, evaluates, and rationalizes in undifferentiated prose. Methodeutics, Peirce's term for the methodology of inquiry, is how to conduct the typed-mode loop well. Encoded as skills, it constructs and maintains the `smem`.

Modes of reason and the irreducible three. Around the act of testing, philosophy of science built an apparatus of real rigor: Bacon's induction (1620), Popper's falsifiability (1934), Meehl's "soft science" critique (1967), Pearl's causal calculus (2009). Justification got its method, every step of it. But it begins one step too late, taking the hypothesis as given and filing its origin under inspiration. The discipline built an epistemology of justification and none of discovery. Peirce alone named the operation, abduction, and was ignored. The harness runs it as a first-class typed mode.

5 Methodeutics, applied

Putting the theory to work means generating hypotheses as typed nodes the harness can test, instead of trusting whatever a model guesses.

The abductive primitive is a diff. Generating a hypothesis means manufacturing a diff: a before snapshot, an after snapshot, and the perturbation that flipped read as figure against the ground that held (Rubin's Gestalt terms). Separation logic already formalized this move as bi-abduction, inferring the invariant frame around a change, and scaled it to real codebases in Facebook Infer (Calcagno et al. 2009; O'Hearn 2019). `inquire` works at the simplest level: one before/after diff, with the frame inferred from the symptom. The extensions to branches and compositional cases are in the lineage appendix.

The "XOR" used throughout is shorthand for that bi-abductive separation: the figure (what the fix must change) held apart from the ground (the frame that stays invariant). Where bi-abduction infers the frame to make a proof go through, the harness fires the same split as a check, computing the symmetric difference against a known-good oracle and keeping only the cases it flags.

Directed graphs as reasoning representation. Pearl 1988 (Probabilistic Reasoning in Intelligent Systems; Bayesian networks as DAGs of dependencies); Pearl 2000/2009 (Causality; structural causal models, d-separation, do-calculus). Pearl's lineage was built for causal-structure inference; our data structure (typed nodes, directed edges) puts it to hypothesis representation. The difference from a Bayesian network is one of kind, and runs deeper than dropped probabilities. A Bayes net conditions over a fixed variable set and propagates probability along edges of dependence; the hypothesis graph abduces its nodes as the inquiry runs, its edges genealogical, a dead hypothesis naming its successor rather than a conditional dependence. A Bayes net is justification over a space it is handed; the hypothesis graph generates the space.

Isn't generating that space just debugging? It is, and debugging has been automated for decades: spectrum-based fault localization, statistical and delta debugging, model-based diagnosis, and search-based program repair are mature fields (Jones et al. 2002; Liblit et al. 2005; Zeller & Hildebrandt 2002; Reiter 1987; Le Goues et al. 2012; Monperrus 2018). Debugging tools already automate the loop; what is new is that the hypothesis graph persists it. Every engineer, and every repair tool, runs some version of abduce a cause, kill it on evidence, witness the survivor, derive the fix. But the engineer runs it in their head, and the tools, whatever logs they keep, do not persist the search as a typed, replayable hypothesis graph.

We implement this loop as a tool. abductor (github.com/kimjune01/abductor) externalizes the diff generation outside the context window, so a model has to represent the rule instead of tabulating the case in front of it: it enumerates a space wider than the model's hypothesis, calibrates each case against a known-good baseline, and exposes one pass/fail gate, with the answer key held outside the model's view. A failing case is a counterexample that forces the next fix, and the search records itself as the hypothesis graph, fixes as nodes and counterexamples as edges.

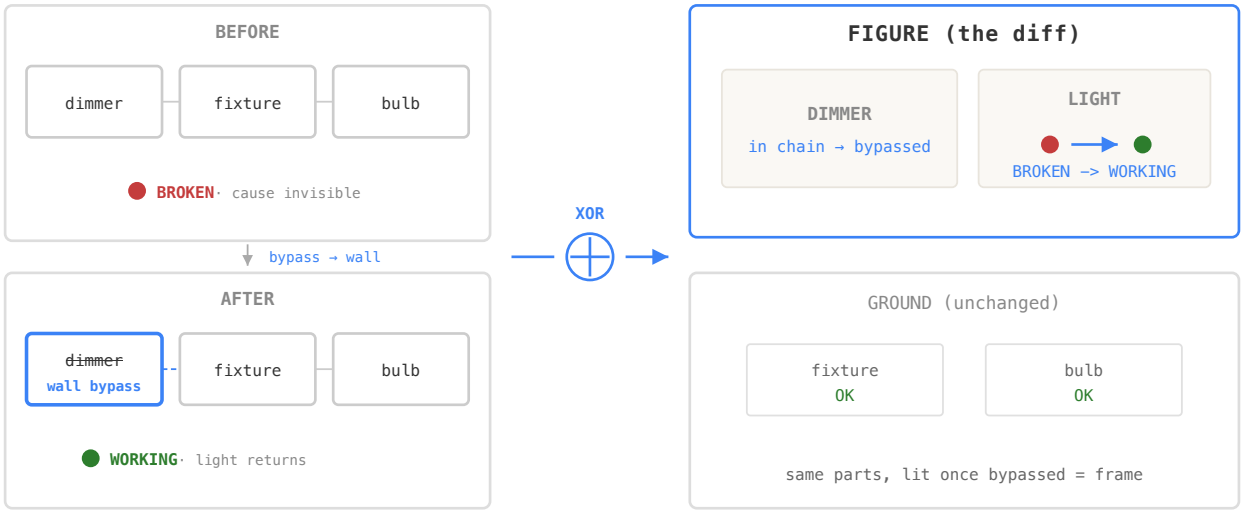

Figure 4: Bi-abduction on a dead fixture. With dimmer, fixture, and bulb all intact, the static scene names no suspect; the perturbation bypasses the dimmer to the wall, and the XOR isolates the figure (the dimmer) from the ground (fixture and bulb).

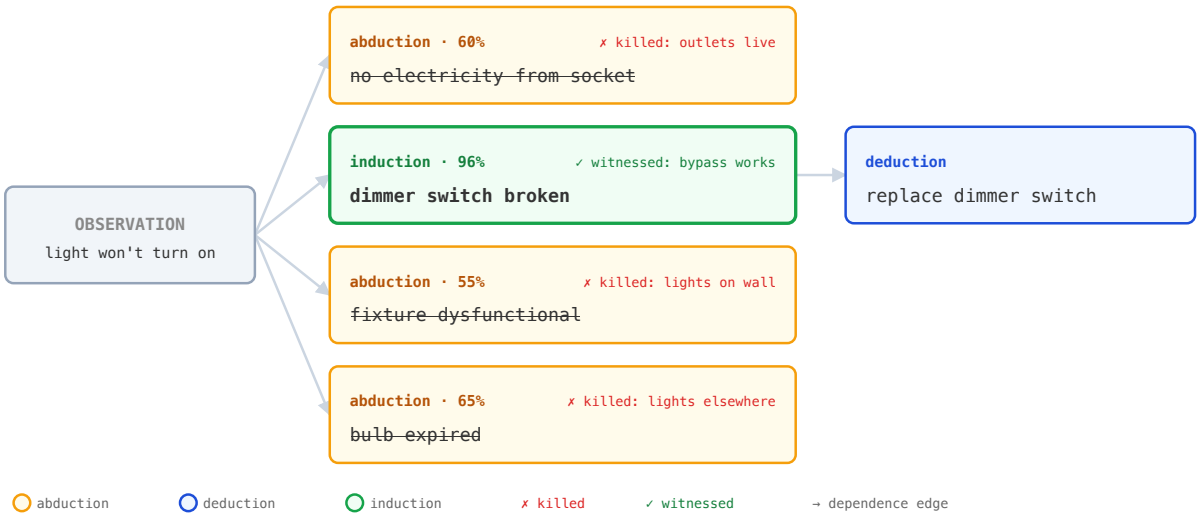

Figure 5: The hypothesis graph for the dead fixture. Abduction fans the observation into four typed candidate nodes; mechanical kill predicates fire on three (the socket, fixture, and bulb each cleared by a cheap test), the dimmer node is witnessed by the bypass and closes the last open hypothesis, and deduction derives the fix. Typed nodes, directed edges, all three modes in one inquiry.

6 The methodeutic harness

How do epistemology and debugging become an agentic harness? The loop that writes the graph needs four things:

- Iteration. One pass can't be trusted, so the loop re-enters on failure.
- Deterministic oracle. A model can't be trusted to catch its own bias, so the check comes from outside the weights.
- Three modes of reasoning. Abduction, deduction, and induction stay typed and separate, each capped at its own confidence.
- Semantic memory. The reasoning is recorded into the hypothesis graph and survives the context window.

Concretely, this is a skill with a tool call in a loop: the outer deterministic driver invokes an agent via the `inquire` skill, who accesses the deterministic `abductor` tool. It mechanizes the XOR operation.

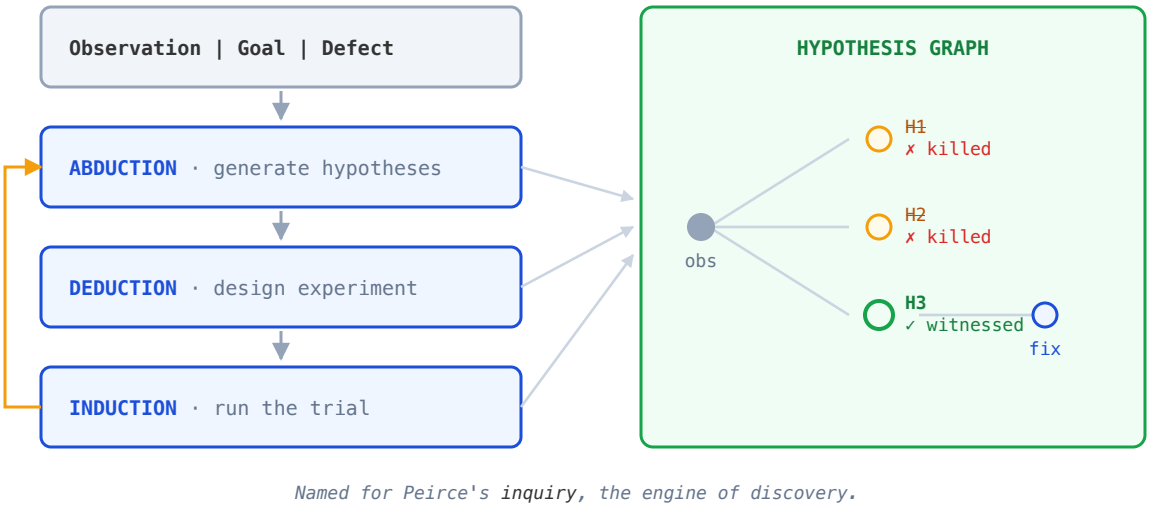

Figure 6: The `inquire` skill: Peirce's three modes as a procedure that writes the hypothesis graph. Induction fires a deterministic kill or witness with no model arbitrating. `implement` and `attest`, which read the survivors and verify the patch, follow below.

6.1 The inquiry frame

We recast each issue as an inquiry on an engineered system: a failure trace, a codebase, a root cause to find, and an intervention that must not regress the rest of the system. Code is the right substrate for the hypothesis graph because it combines three properties that other inquiry domains rarely bring together:

- Reproducible: same input yields same output, modulo controlled nondeterminism
- Deterministic: causal lines from input to behavior are mechanical
- Perturbable: single-line and single-function diffs are cheap to apply and fully observable

Because those three hold together, kill conditions over code are exact executions.

One trial settles the predicate in this regime. In code the per-case response is mechanically observable, so a single passing test on a captured diff is a complete verdict that the diff satisfies the executable predicate for that case. That verdict speaks to the predicate alone: behaviors it doesn't cover are out of scope, a boundary that §7 shows is exactly where the differences live. Where such verdicts are aggregated, the right summary is counts and denominators rather than confidence intervals: per-case verdicts are exact, and aggregating them is bookkeeping.

The three Peircean modes are how `inquire` builds the graph, each node typed by the mode that established it and capped at that mode's confidence:

Mode	What `inquire` does	Confidence
Abduction	Proposes candidate root causes from the observed failure; writes hypothesis nodes with falsifiable predicates and kill conditions (read-only)	low
Deduction	Traces each hypothesis's consequences through the code to localize the suspect set	high
Induction	Tests survivors with cheap read-only experiments (prints, intermediate data)	moderate

`implement` then writes the surviving hypothesis, with an adversarial challenger critiquing the diff against the spec. `attest` runs the test suite, takes the grader's pass/fail verdict, and emits a re-entry route (`inquire`, `implement`, or `none`) from a fixed verdict→route table. The driver parses the verdict and the route; both are mechanical, and no model decides termination.

6.2 Hypothesis graph output

`inquire` emits the hypothesis graph: the structured-analysis document that precedes the patch. Kill conditions are mechanical predicates over the evidence trajectory, so a node dies when its predicate fires and not before. The graph persists across iterations; re-entry adds nodes rather than overwriting. The frontier closes only when every open hypothesis is killed (a test refutes it) or witnessed (a test confirms it).

A committed node is a conclusion, and an inquiry that reaches one rarely runs straight. Following the `inquire` skill on a real bug, a single hypothesis flips across all three modes and a kill before it settles:

> abduction → deduction → kill → abduction → deduction → induction → deduction → induction ⇒ induction

An in-flight inquiry trace, illustrative: Sonnet 4.6 following the ***`inquire`*** skill on the python-dotenv ***`find_dotenv`*** v1.0.1 regression (a real, reproducible bug, every command run). The active hypothesis cycles through all three modes and a kill before the inquiry settles; a committed graph records only the terminal node (induction) and discards this sequence. Full trace: recon-inflight-dotenv.md. Not a frozen Pro instance.

6.3 Deterministic gating

The control loop is standard: the driver routes on `attest`'s verdict and re-entry route under a bounded attempt budget, and a failure re-enters `inquire` with the updated graph rather than retrying the patch, so the hypothesis graph doubles as the loop's checkpoint and no dead branch is re-proposed. That leaves the inner layer still owed a demonstration. The oracle the opening made the crux is what the gate leans on, and §7 isolates it on a single bug, where it does most of the work the bench number seems to credit to diagnosis.

6.4 Artifact availability

All code and data are openly available, each developed in a public repository and archived under a DOI for permanence. `abductor`, the tool that mechanizes the gate, is developed at github.com/kimjune01/abductor and archived at doi.org/10.5281/zenodo.20738162 (v1.0.0, AGPL-3.0). The mechanism experiment of §7 is developed at github.com/kimjune01/hygraph-mechanism and archived at doi.org/10.5281/zenodo.20754118 (v1.1.0, CC BY-SA 4.0); the archive carries the preregistration, the rebuild-confirmed dataset, the regrade script, and the climb traces. Every reported result reproduces from the committed inputs of the archived version.

7 The mechanism by case study

What would prove that we had encoded reasoning into the harness, rather than just prompted a model into a better answer? The signal has to be one prompting alone can't produce:

- Without the reasoning operation, the bare model can't reach the fix.
- Handed the operation, it reaches the fix.
- The fix generalizes to cases it was never shown.

Here we demonstrate an implausible result on a single bug. Even Fable, the strongest released model, could not resolve it on its own; the methodeutic harness carried Sonnet 4.6 to human-level completion.

7.1 Coding benchmarks measure translation ability

We first went looking for that signal on an established coding benchmark, and learned such benchmarks aren't built to show it. A SWE-bench-shaped coding benchmark hands the solver three things: the issue

text, the repository at the buggy commit, and a hidden test the fix must turn from red to green. The issue text often hands over the specification, removing the need for a discovery step: finding the root cause and the general fix.

SWE-bench Pro (Deng et al. 2025) is the dominant such benchmark, the one OpenAI now recommends over Verified. For a competent model, the well-specified tasks are one-shot from the prompt, so there is no cause to discover. The underspecified ones are immune to discovery because the author's intent is hidden from the issue and repository. Our determinacy audit of all 728 public tasks measures this (audit).

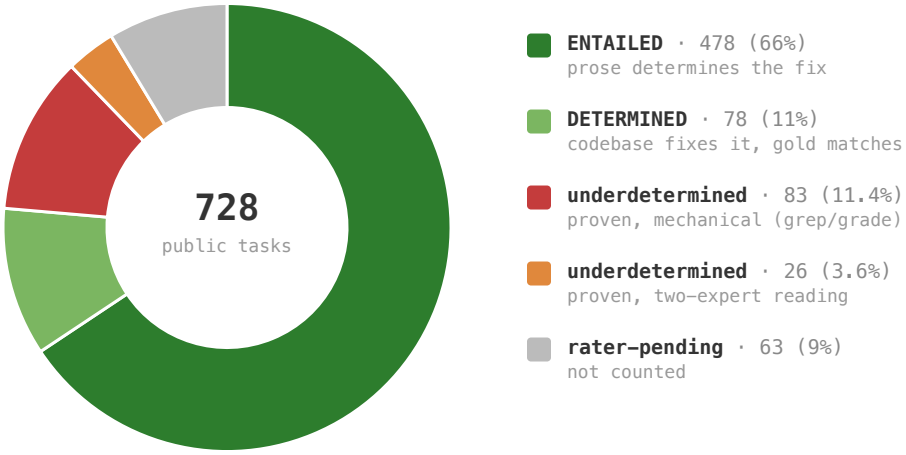

Figure 7: SWE-bench Pro's 728 public tasks by determinacy. The green majority is one-shot, with nothing to discover; the 109-task underdetermined floor (15.0%) grades the author's unstated intent, undiscoverable from the materials.

With the tests as oracle, the harness resolves 95.3% of SWE-bench Pro's public split under the official grader (bench run): the specification-to-implementation translation step is largely solved. So the interesting work is now discovery: a bug with a deep, discoverable cause. This is most of software left to do, now that coding agents dominate implementation.

Moreover, Pro's public set carried a risk we were not willing to take: contamination. So we went looking for a post-cutoff bug where discovery is the whole difficulty. Verus #2219 is one: a March 2026 issue, opened and fixed after the solve models' training cutoffs, so the case is contamination-free. The maintainer's narrow fix (PR #2230) passes its shipped test, and a general fix eventually landed too (PR #2501, merged 2026-06-05). The experiment is openly reproducible on GitHub, with further analysis within.

7.2 Verus #2219

The bug is verus-lang/verus#2219: ghost uses of never type invalidates borrowchecking.

Verus is a deductive verifier for Rust: you annotate a program with specifications and ghost proof code, Verus discharges the obligations through the Z3 SMT solver, and a passing run certifies the code meets its spec. The cardinal property of any verifier is soundness, that it never certify a program which violates its spec, because every proof built on top inherits that guarantee. An unsoundness, where the verifier blesses a program it owes a rejection, is the worst defect it can carry and the hardest to notice, since the symptom is silence. Nothing fails.

The correct fix must range over a whole class of program inputs. Two of them look identical at the `!` token and owe opposite verdicts:

```
proof fn unsound(tracked s: S) {
    ghost_terminal();  // "I STOP here!" ...a lie, proof-only, deleted before the program runs,
```

```
    eat(s);             // so... oops! these two lines actually DO run. Verus thinks they don't,
    eat(s);             // but `s` was already spent. OH NO: bad code waved through. THE BUG.
}

fn sound(tracked q: Q) {
    real_terminal();              // this one really does STOP here (CRASH!)
    proof { take(q); take(q); }  // so these genuinely never run, safe to skip. All good!
}
```

Identical at the token, opposite at the root: the contrast the experiment needs. The burden of telling them apart falls upstream, which makes the check stateful and forces it to be exact. Other reasons it is a good bug to study:

- The maintainers were genuinely stuck, and the merged general fix took deep Verus knowledge and three months. Not trivial.
- The symptom sits far from the cause: from the wrongly-accepted program, the fault sits four steps up the chain (§7.3), so localization has to climb to find it.
- The project's own suite passes for both the narrow and the general fix, so it cannot see the distinction the fix has to make.

7.3 Diagnosis

A ghost expression of type `!`, erased before compilation and so not actually diverging, still triggers rustc's never-type edge prune: Verus drops a live control-flow edge, skips the reachable code behind it, and verifies a program it owes a rejection. That is the unsoundness.

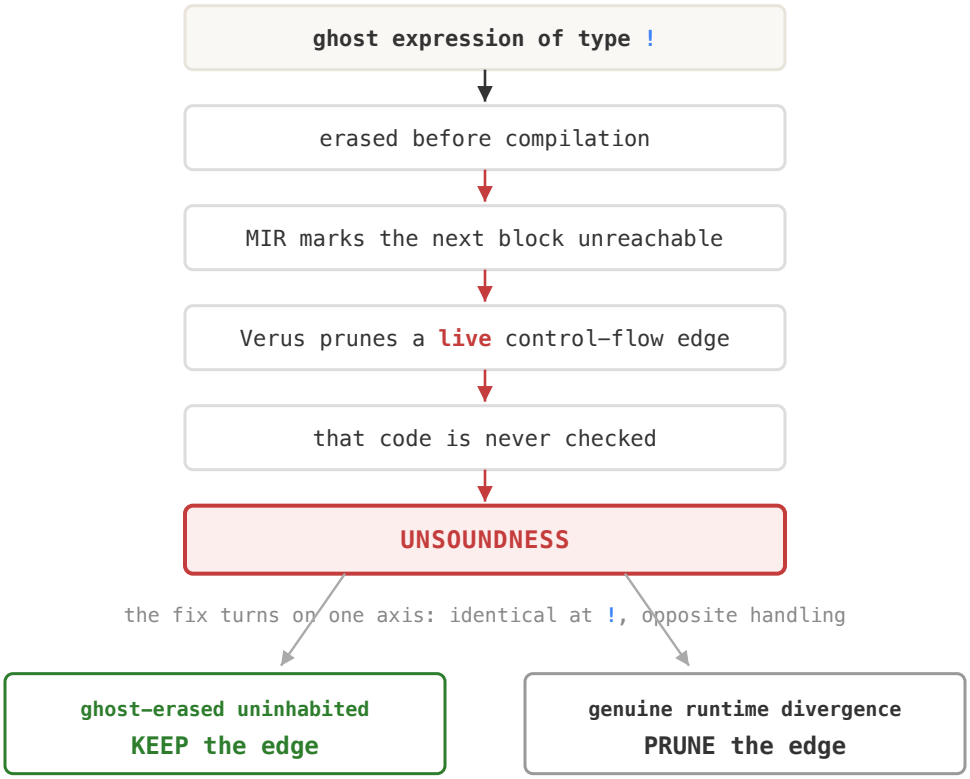

Figure 8: The #2219 unsoundness as a causal chain. The axis at the bottom is the crux: a ghost-erased uninhabited return (keep the edge) and genuine runtime divergence (prune) are identical at the token, opposite in handling.

The narrow fix keys on the surface token: when the diverging expression is a literal `!`, keep the edge. The maintainer's first patch (#2230) does exactly this, and it is the ceiling every self-graded method reaches. The general fix keeps the edge for the whole class of uninhabited types the `!` token never names, which a surface-token fix cannot reach. That distinction is the XOR the experiment lives on.

That diagnosis was not handed down; an inquiry climbed to it, and the climb is retained:

> H0 · observation (induction). `reported.rs` rejected pre-patch with Rust E0382, "use of moved value". Edge: is E0382 incidental? Vary the tracked type's copyability and the never-expression shape.
>
> H1 · the erased `!` is load-bearing (abduction). Trial `copy_int_never.rs`: erased `!` before a double-used Copy int. Observed: "3 verified, 0 errors", then a rustc ICE, "bad arg #0 (`!` ← `()`)". Verdict: the erased `!` corrupts MIR even where E0382 cannot fire. Trajectory: convergent on the erased ghost never-value.
>
> H2 · the source expression is incidental (abduction). Trial `never_let_binding.rs`: `let x: ! = arbitrary(); test(x)`. Observed: rejected, "never-to-any coercion not allowed in spec mode". Verdict: the explicit-coercion path is already covered; the uncovered path is the direct erased call in argument position. Kill: "all never coercions are uncovered."
>
> H3 · root cause (deduction). Spec calls erase to `erased_ghost_value` at the original Rust type; when that type is `!`, the injected verifier-only value is never-typed and breaks the MIR path.
>
> H4 · fix (deduction). Reject erased ghost calls whose adjusted Rust type is `!`, before erasure reaches MIR construction.

One inquiry into #2219 (Sonnet 4.6), condensed from the full record. It reached the surface-token narrow fix (the #2230 ceiling) and stalled at a candidate it could not rebuild to verify. One markdown file per inquiry; the runs left many: pilot 11.

7.4 Experiment design

To demonstrate the mechanism, we hold one reproducible control and ablate the factors for attribution. Estimating how often the mechanism is responsible for the outcome is a separate, statistical question, out of scope here (§8.1 bounds where it engages).

Here, we demonstrate two ablations of interest:

- Verification mechanism. How each kill gets its verdict: by the model's own attestation (self-attested) or against a reference the model cannot author (externally verified). External means external to the model; this axis is defined in §6.
- Methodeutic inquiry. Whether the run runs the typed three-mode loop, and at which tier abduction is applied: written into the prompt, or lifted out as a tool call (`abductor`). The baseline leaves it to a bare prompt with neither.

Six methods self-attest, one is externally verified, ordered from least to most intervention, with minimal and neutral sharing a row (identical settings, both run). Enumeration here is breadth of search rather than one guess, over suspect code sites (the site-enumeration arm) or a wide space of discriminating test programs (● present, ○ absent):

Experiment arm	Enumeration	Kill conditions	Externally verified
minimal/neutral prompt	○	○	○
site-enumeration prompt	●	○	○
abduction prompt	○	●	○
hypothesis graph prompt	○	●	○
self-verifier harness	●	●	○
abductor-enabled harness	●	●	●

Each arm writes a hypothesis graph by the same loop and runs three times. (Oracle here is metrological: an instrument checked against a reference, distinct from the probability-calibration of confidence scores.) We drew candidates from the localization-hard band the lead case sits in. Before any arm runs, the protocol preregisters one sentence per loop: testing X, predict Y, refuted by Z.

For ecological accuracy, each model resides in its own native agent rather than a uniform rig we impose: Claude models in Claude Code, GPT-5.5 in codex, Composer 2.5 in Cursor. Our harness is a thin wrapper that drives each vendor's CLI non-interactively with the same stage prompt:

```
claude -p …        # Claude Code
codex exec …       # codex
cursor-agent …     # Cursor
```

The meta-loop owns only the stage contracts, the hypothesis graph, and the gate; every model-facing call goes through the vendor's own agent. Full configs, prompts, and per-arm logs live in the mechanism repository.

But external verification is necessary. It is not sufficient. A reference only helps as far as its cases cover the XOR the fix must draw (§7.3), so what matters next is where those correct verdicts come from and which side of the distinction each one reaches.

7.5 Making a tiny bench

To attribute the lift to the mechanism, every arm has to be scored on the same mechanical ruler; otherwise "more general" stays an impression no one can check. The project's own suite is no such ruler: it passes for both the narrow and the general fix (§7.2). So we build one by hand, a tiny answer key. The case is one bug, so the bench is one bug. It holds three compiler states, the buggy commit (base) and the maintainer's narrow and general fixes, plus a handful of small test programs, the probes. A correct compiler owes each probe one correct verdict: `REJECT` for an unsound program, `ACCEPT` for a sound one. Grading a fix is then mechanical: does it return each probe's correct verdict?

Probe program	Golden assessment	Who supplied it
bug probes (uninhabited)	`REJECT`	base, by construction (free)
divergence, in-bar	`ACCEPT`	the merged human fix (#2501)

The split is the whole asymmetry. Bug probes are uninhabited by construction, so their correct `REJECT` is free and base is wrong on exactly them. The gate (§7.7) enumerates such cases and passes a fix only when it flips them all to `REJECT` with no sound case newly rejected, no human in the loop. Divergence probes are the hard side, sound and to be preserved, but base is no reference there and the gate's grammar holds no divergence-preserve shape, so the gate is blind to them. Their `ACCEPT` comes from a human, the merged fix #2501 (with one stretch case even #2501 declines), and stays held out of the gate. Passing it therefore tests whether a fix represents the rule rather than fits the gate. The easy side's verdict is free; the hard side's costs a human (§7.8).

A recall probe rules out memorization: asked in isolation how #2219 was fixed, the model does not recover the patch, so its solution is reconstruction. Every verdict is from a forced-fresh, fingerprinted build; the frozen dataset and regrade script are committed.

7.6 Results

Which arm climbs past the narrow fix? The six self-attested methods plateaued on all eighteen draws, three each; only the externally-verified arm broke past.

Arm	Gate pass	Cases flipped	Outcome
minimal / neutral	✗	114	narrow plateau (#2230 slice)
site-enumeration	✗	114	narrow plateau
abduction	✗	114	narrow plateau
hypothesis graph	✗	114	narrow plateau
self-verifier	✗	114	narrow plateau
`abductor`	✓	269	general on the bug-set, wide-but-broken on divergence

Cases flipped is the modal of three draws; gate pass is the gate's verdict on its own cases.

The six self-attested methods all stopped at the narrow plateau, none reaching `pass=true`. Only the `abductor` arm broke off it, with zero valid-preserve rejections on the gate's own cases. Its fix is more general than the plateau but not yet the merged human fix: it stays wide-but-broken on the in-bar divergence case the gate never enumerated (§7.7). The reach is real: it flips the whole bug-set and rejects two out-of-grammar held-outs the gate never showed it, applying its general predicate to cases it never saw.

That isolates the mechanism on one model. The headline lift is a slice across two: the same bug handed to the strongest released model without the abductor, and to a weaker one with it.

	No abductor	+ abductor
Fable 5 (strongest released)	wide-but-broken	general fix (#2501)
Sonnet 4.6	narrow (#2230)	general fix (#2501)

Two contamination-clean models, one harness. Without the abductor neither reaches #2501, and they miss in opposite directions: Fable too wide, Sonnet too narrow. With it (§7.8) both land the human fix the stronger model cannot reach on its own. One draw per cell.

7.7 `abductor`

`abductor` is an open-source command-line tool anyone can run (github.com/kimjune01/abductor): the abductive XOR of §5, built as a standalone instrument on the program-analysis lineage that put abduction to work (§15). On this bug it carried a model past the plateau prompting alone could not clear. Our read of why is that it externalizes that XOR, the symmetric difference between what the model believes and what is true, from the inquiry loop into three domain-general operations with no answer built in:

- It enumerates a space of cases closed under the property's type-formers, wider than any one hypothesis.
- It calibrates each case against a known-good baseline, the comparator the model cannot author, so the ground truth is external to it.
- It gates on a single pass/fail signal the model iterates against.

The prompt demands generality and supplies the signal, so the rule is the model's own reconstruction. Handed only the instruction to range over type-formers and judge, Fable rebuilt ***`abductor`*** itself from the goal alone (§7.8): its own 7,026-case gate over the uninhabited type-formers. A general construction permits that; a bespoke one, with the answer baked in, would not.

The gate's coverage is the frontier, lever and limit at once. Where it covers, the uninhabited side, the model greps the codebase and widens its predicate to clear the cases the gate feeds, supplying the discovery while the gate gives direction. Where its grammar is silent, the divergence side, it over-generalizes: the fix lands general on the bug set and over-conservative on divergence, wide but broken, the §7.3 XOR collapsed to an OR where the gate stayed silent. The lift it buys is a coarse mode gate. It is no recovery of the verifier's decision procedure: a fix keeping the edge for every ghost-mode call, with no inhabitedness query at all, grades identically on every probe. Supply the missing divergence verdicts (§7.8) and the corrected arm reaches the human fix.

The construction generalizes past this bug, and past verification: the same three operations reconcile any two accept-sets, a refactor against its original or a port against its source. They recover the disagreement from a compact sketch, so neither set need enter a context window or cross a wire. The full command surface and the cited lineage are in the repository.

7.8 Abduction is unavoidable

A model can bootstrap the enumeration, the combinatorial breadth, because that is mechanical: handed a vague prompt and no gate, Fable rebuilt one for itself (§7.7). It cannot bootstrap the correct verdicts, the human-accepted answers. Its self-built gate labels each case with the very predicate under test, so the one case that needs a truth from outside its own belief, genuine divergence versus ghost-erasure, gets self-mislabeled as handled. One run shows both halves: a wide net the model built, and a blind spot exactly where its own labels could not reach. The claim is scoped to that self-grading circularity. It is not a proof that no model could ever produce a correct verdict some other way.

So before crediting the verdict source, the cheaper explanations have to fall, and each has a control already in the ablation.

Did the abductor just search harder? The self-verifier searched exactly as hard: handed the same prompt and no tool, it built its own case generator and brute-forced some 7,026 cases a round, the breadth the abductor enumerates, and it still plateaued: confirmation bias (§4) with a receipt, grading against its own belief, the one move the external comparator forbids. Enumeration sat on both sides of the comparison, so it is not the lever. Nor is reasoning depth: the abduction arm ran textbook bi-abduction to the semantic predicate the general fix turns on, named it in prose, self-validated it against its own programs, and still plateaued at the same narrow verdict as the one-line minimal arm. An arm can write down the correct predicate and certify it against its own belief; the patch ships narrow all the same. The two-model rerun sharpens the point: without the abductor the models miss in opposite directions, Fable too wide and Sonnet too narrow, and the abductor pulls both to the same #2501. More search does not correct two opposite errors onto one target; an external answer key does.

Was the model handed the fix? It was handed correct verdicts on cases. It never saw the patch, the predicate, or the Rust API. The comparator returns only pass/fail on candidate behavior, so the model still localizes the fault four steps up the chain (§7.3) and writes the fix itself. This is why the signal is verdicts and not the issue text: where a benchmark leans on the issue to say what fixed means, and hands over the spec doing it, the gate says it in correct verdicts on cases, the more specific signal, grading whether the fix draws the exact behavioral boundary rather than whether it parsed a prose description.

A better scaffold, then? The self-verifier is the scaffold control: the whole loop, enumeration and kill conditions included, with the external verdict the only piece removed. It plateaus beside the bare prompt. A scaffold without the answer key buys nothing here.

Just the stronger model? The attribution rests on the within-model contrast, where the model is held fixed and only the verdict source moves. The cross-model grid (§7.6) only illustrates the consequence; the within-model ablation is what proves the cause. It shows the lift is wide enough that a weaker model with the verdicts passes a stronger one without them.

The whole ablation reduces to one causal diagram: across its arms, everything stays fixed but the verdict source, the one input the model cannot author.

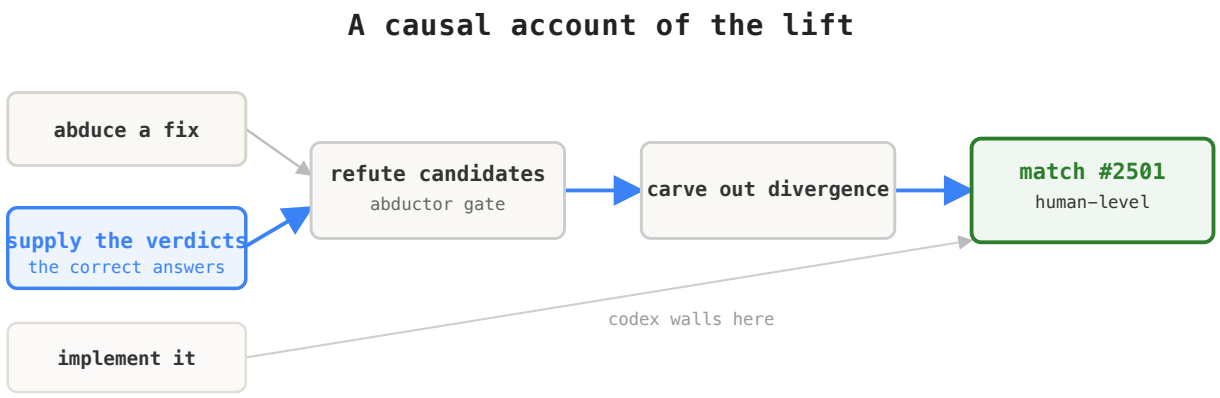

Figure 9: Reading the diagram: each box is an action, read left to right. Four factors stay identical across the ablation's arms (model, loop, bug, graph), so none of them can be the cause; only the verdict source varies. Gray boxes the model can build for itself; the blue box is the one input it cannot author, and the blue path is the causal route. The gray arrow into the outcome is the implementation caveat, where codex walls.

Not one model's artifact. The §7.6 lift is one model pair; swap in the corrected gate, verifying against the correct verdicts (#2501), and the same fix lands across four models in four native CLIs:

Verdict source	Narrow fix (#2230)	General fix (#2501)
self-attested, six methods	•	○
externally verified — Fable	•	•
externally verified — Sonnet 4.6	•	•
externally verified — Composer 2.5	•	•

Verdict source	Narrow fix (#2230)	General fix (#2501)
externally verified — codex	•	○

codex clears the bug arm but walls on the divergence case, an implementation limit the verdicts do not remove.

The convergence is coverage-bound: it shows the corrected-gate fix is reproducible across workflows. It does not show three models independently rediscovered the predicate. The shared decline on the stretch case is most likely the gate funnelling every successful arm into the one behavior it rewards, which the human fix happens to share.

Retrospectively the correct verdicts are free; prospectively, on a fresh bug with no merged fix, the easy side has one and the hard side does not. The division of labor is the deployment design: the model enumerates and fixes, the harness draws its verdicts from approved history, and the residual hard side gets named. Nothing is hidden.

7.9 Abduction becomes a tool call

Step back from the arms. The surprise the traces catch is the models performing the XOR by hand: the self-verifier recomputed the full symmetric difference every pass (§7.8), the tell that the XOR is the operation abduction runs on. But a model grading its own cases can only check them against its own belief, so it plateaued. The model had the idea all along; what it lacked was a place to run it.

That is the signal we went looking for. The warrant-producing core of abduction is a mathematical operation, the symmetric difference between what the model believes and what is true, and in coding diagnosis it lifts off the model and runs as a tool call, the one we release as `abductor`: enumerate, calibrate, gate. The leap stays creative and the check turns mechanical, the division of intellect the introduction set out to buy, and the run leaves behind the hypothesis graph the introduction promised, written by the experiment that needed it.

8 Discussion

8.1 Diagnostic frontier

Implementation is the solved part. A competent model turns a well-specified issue into a passing patch, which is most of what a SWE-bench task rewards (§7.1) and most of what coding agents already do. What is left is discovery: the root cause when the symptom sits far from it, the general fix when the suite cannot tell it from a narrow one. That diagnostic step is the frontier, and it is where the mechanism engages.

The band is narrow precisely because implementation is solved, so most bugs never reach the diagnostic step. On pilots from the merged-PR pool and beyond, a minimal baseline solved them unaided, from trivial fixes (qrtool #695, bat #3710) to the cases that most invited the machinery (slang-server #310, fjall #287). Two findings attribute the nulls. A selection artifact: the deployment pipeline's own triage fast-paths easy bugs around the graph ("a 1-line fix with a confirmed reproducer doesn't need a hypothesis graph"), so the merged pool is exactly the implementation-solved subset. And baseline reach: a capable model in a minimal loop fixes most reproducible bugs unaided. The band is bounded below by that reach and above by triage, which is why a population rate over an unselected pool measures the band's width and misses the mechanism; the Verus case is chosen to sit inside it. These nulls publish attributed, each arriving with the mechanism that produced it, so an unexplained null says stop where an attributed one says where to point the next instrument.

Automation is a different axis. A faster loop, a parallel fan-out, a cheaper scaffold moves the same work along quicker, and none of it changes whether the agent can find the cause. The lift here is on the diagnostic axis, the distinction the scaffold-synergy results blur (§9.2): a weaker model with better plumbing finishing sooner is a throughput gain, where a weaker model reaching a fix a stronger one could not is a capability gain.

8.2 Verifiable agents

The reframe is from treating an agent as a source of verdicts to a source of claims bound to evidence.

The failure mode is output that is confidently wrong and not cheaply verifiable, and it scales with model capability rather than against it, the default product of fluent generation. The Verus self-built gate is the instance (§7.8): green by the agent's own check and unsound at once, exposed only by the held-out

probes replayed against the build. The same shape recurred throughout, in the wide-but-broken fix, the 4/4 false-green self-audit, and the recall-inflated resolve rate.

The substitution is accountability in place of trust, checkable line by line. The self-graded agent produced a verdict, its own gate green; the externally-oracled inquiry produced a record of typed nodes, each a hypothesis, an exact command, an observed outcome, and the edge the result generates, every consequential one replaying on a clean build. Such a record is audited step by step without extending trust to its author, and the better fix is identifiable independent of the system that produced it. That property, rather than the outcome of any single comparison, is what survives.

Truth is buildable. The node semantics of §3 is what gives the trace its force here. If a true claim is a build presently passing and its warrant lives in the edges, then a node without a replayable trial is not a node at all, as an uncheckable number is not a measurement. A resolve rate, however honest, is a verdict over 728 builds an auditor cannot inspect; a hypothesis graph is the inspectable chain.

The cost asymmetry of fabrication. A fabricated reasoning trace is expensive to sustain, because every fabricated node must survive a replay the author does not control. A confident narrative is cheap to produce, because nothing in it is bound to a procedure. A self-graded method's over-narrow fix is the cheap kind: it reads as complete, and closer reading does not reveal the flaw, while the receipt reveals it in one command. Verification is therefore part of the method rather than overhead on it. The same asymmetry governs failure: a trusted verdict that fails leaves nothing behind, whereas a recorded inquiry that fails leaves a trail naming the failed node. The Verus climb recorded its own corrections in trail, the narrow `is_never` fix refuted by the very cases the gate fed it, each a kill that generated the next edge and itself a replayable trial.

Hidden effort is not reasoning you can check. The frontier labs ship "reasoning" as an opaque dial (`high`, `xhigh`, `ultrathink`): more private tokens burned before the answer, the model grading against its own belief, harder. The Verus result is where that runs out (§7.8). Effort scales the inducible half, the enumeration, and cannot manufacture the half that is not, a ground-truth oracle from outside the model's belief, so the self-verifier plateaus where a single externally-graded kill reaches the general fix. Whatever the dial buys ships nothing a stranger can rerun; an effort setting is the purest take my word for it, a larger number where a kill condition should be. The position here is the opposite: put the reasoning in the harness, typed and replayable, its level a trail the auditor checks instead of a knob the auditor trusts.

Accountability, in deployment. The intro's promise was a claim that clears only when an independent party reruns its recorded trial (§1); the receipts deliver it. To a maintainer the agent and its operator are strangers, and with the trial attached that stops mattering: a red-on-master/green-with-fix test and a clean suite read identically whoever submitted them. This is the review bottleneck answered by making the check cheaper than reconstructing the reasoning.

Merit attaches to the work, not the doer. An agent can produce many artifacts of any quality, so judging them by their author is backwards; judging the author by replaying the artifacts is the only direction that scales. Merit is the warrant a piece of work carries in itself, checkable without reference to who or what produced it: the hypothesis graph is a unit of work that ships with its own evidence, the receipts-first PR the corresponding practice, the maintainer who merges on the ledger alone an early instance.

9 Related work

9.1 Construct validity and contamination

The SWE-bench family defines the Verified / Pro lineage, official harness, and contamination-resistant tier design. SWE-Bench+ (Aleithan et al. 2024) manually audited the original bench: 32.67% solution leakage, 31% weak tests. OpenAI's February 2026 audit found a majority of audited Verified tasks have flawed tests and that frontier models reproduce exact gold patches; it stopped reporting Verified and recommends Pro. Wang, Pradel & Liu (ICSE 2026) show plausible patches pass tests yet diverge from developer intent; their axis is patches that pass but are wrong, ours (§7) is tasks whose materials do not determine which passing behavior is intended. ORACLE-SWE (arXiv:2604.07789) quantifies the same handover, ablating the oracle and specification signals that leak through a task and measuring the resulting drop; SLUMP (arXiv:2603.17104) opens on the identical premise, that benchmarks supply the full specification upfront while real coding does not, and answers it by building an underspecified-by-design benchmark. Here we part from both on the verdict: they treat handover as a defect to fix with a better benchmark, while the determinacy audit (§7) draws it as a category boundary, a spec-conformance instrument cannot be tuned into a measure of diagnostic inquiry because the two are different types. Evaluation surveys reach the same edge from the methodology side: scoring final outputs misses the in-run reasoning and failure causes, and

trajectory-level assessment is the called-for remedy (Yehudai et al. 2025), which is what grading the trace rather than the patch (§11) would supply. SWE-rebench uses post-cutoff filtering as a parallel contamination strategy; LiveCodeBench (Jain et al. 2024) is the origin of post-cutoff (temporal-holdout) evaluation, and the standard objection to it applies here too, that training cutoffs are porous because RL post-training and inference-time retrieval can surface later content. The witnessed case (§7.5) is built against exactly that objection: it is post-cutoff and the solving model's weights predate the fix, so neither porosity nor retrieval supplies the answer. HAL (Stroebl et al. 2025) sets the cost-transparency precedent and the official swe-bench/experiments repo the per-instance publication norm (`trajs/logs/patch/report`); the receipts here extend both with gate traces, hypothesis graphs, and a re-gradeable cost ledger.

9.2 Agent scaffolds and SE-agent harnesses

Surveys and position papers now map the harness's responsibilities directly: the case for programming with trust, verification and analysis built into the agent rather than retrofitted after (Roychoudhury et al. 2025), the systematizing of agents for software engineering around verification, testing, and repair (Liu et al. 2024), and the finding that current agents lack the persistent, structured memory long-running work needs (Wang et al. 2025). We answer with the structure none of them name: a Peirce-typed, kill-conditioned hypothesis-graph memory. The SWE-bench-targeted harnesses that exist, OpenHands (Wang et al. 2024), SWE-agent (Yang et al. 2024), and AutoCodeRover (Zhang et al. 2024/25), are ReAct-pattern loops (Yao et al. 2023) without it. Voyager (Wang et al. 2023) is the closest loop-shape precedent: embodied observe→hypothesize→test→commit, with a skill library where this work holds falsifiable claims. SWE-Effi (arXiv:2509.09853) is the sharpest published counter-position: effectiveness emerges from scaffold-model synergy rather than residing in the scaffold alone. Here we agree from the other direction, with the synergy named: the binding pair is gate × oracle, isolated in the controlled experiment of §7 rather than on a benchmark that cannot see it.

Recent results do report a weaker model with a scaffold beating a stronger one, but only on the automation axis; the capability axis is untouched, and both claims dissolve on a close read. GradleFixer (arXiv:2510.08640) has Gemini-2.5-Flash out-repair Gemini-2.5-Pro on Android builds, one generation apart inside a single family; the lift is domain-specific execution primitives for Gradle, and a model handed purpose-built tools for one build system doing better is the expected result. Confucius Code Agent (arXiv:2512.10398) has Sonnet 4.5 edge Opus 4.5 on SWE-bench Pro, 52.7 to 52.0, but the edge reverses the moment Opus runs the same scaffold (54.3), which marks it a scaffold effect rather than a capability crossing. Both externalize actions or notes; the diagnosis stays in the model.

The lift in §7 parts from these on three counts: it is a frontier extension rather than a throughput gain; the externalized operation is domain-general (the released `abductor` carries no answer and no domain model, §7.7) rather than a tool shaped to one build system; and a controlled ablation pins it to the verdict source alone, so it does not reverse when the stronger model is handed the same harness, the way a whole-scaffold swap does. The nearest comparator precedent, SWT-Bench (arXiv:2406.12952), grades generated tests by fail-on-original then pass-after-golden-patch, the same external-golden shape, but as an evaluation oracle for test generation. Ours is the runtime instrument an agent iterates against with the answer key hidden.

Two concurrent developments arrived independently at adjacent points, each carrying one of the two components composed here. Theorem-of-Thought (Abdaljalil et al. 2025) types reasoning into abductive, deductive, and inductive specialist agents per query: the typed cycle, without a persistent typed memory across cycles. Cognitive Memory Manager (Khalid & Arora 2026) extracts a typed-node DAG by observing agent execution and mines it for patterns to promote to skills: the typed graph, mined descriptively where ours is generative (it routes the run). That convergence is independent. Provenance for the framing here is timestamped on the project blog (The Hypothesis Graph, Evidence has a trajectory).

The trichotomy these siblings reach for is Peirce's (1878, 1903), used without attribution: Theorem-of-Thought builds abductive, deductive, and inductive agents while naming no pragmatist in its references. §4 and §15 wire the vocabulary to its sources, and the dated posts above timestamp this lineage's use of the hypothesis-graph primitive.

System	Domain	Reasoning-mode typing	Persistent structure & update	Termination gate
Voyager (Wang et al. 2023)	Minecraft	None	Skill library; test-validated graduation	Test-pass on skill

System	Domain	Reasoning-mode typing	Persistent structure & update	Termination gate
IDEA (He et al. 2025)	Interactive rule learning	Peirce-cited, agent-level	Working rule set	None explicit
ADI (Gilda & Gilda 2026)	Algebraic invariants	Peirce, layered (L0/L1/L2)	Symbolic knowledge graph	None explicit
AriGraph (Anokhin et al. 2024)	TextWorld	None	Knowledge graph (entities, relations, episodes)	None explicit
CausaLab (Yang et al. 2026)	Causal discovery	Causal-typed (SCM)	Evolving structural causal model in a DSL	None explicit
BeliefMem (Liao et al. 2026)	Partial-observability QA	None	Candidate set; Noisy-OR probabilistic update	Probabilistic threshold
Theorem-of-Thought (Abdaljalil et al. 2025)	General reasoning	Abduction / deduction / induction, agent-level (no Peirce cite)	Formal reasoning graph	NLI-guided Bayesian coherence
CMM (Khalid & Arora 2026)	SE (coding agents)	7 trajectory roles, extraction-time	Typed DAG; confidence decay	Human approval + retrieval-validated threshold
This work	SE (industrial code)	Peirce, enforced at write time per stage	Hypothesis graph; mechanical kill predicates on the audit verdict	Deterministic finite-state

Table 1. Comparison spine for adjacent typed-reasoning and graph-memory LLM-agent systems. Cell terseness is by design; prose nuance in §9.3.

9.3 Typed reasoning and graph-structured memory

IDEA (He et al. 2025, ACL Findings, arXiv:2408.10455) explicitly cites Peirce and uses the three modes in an interactive rule-learning benchmark. ADI (Gilda & Gilda 2026, arXiv:2604.15727) gives an explicit Peircean tripartite protocol with epistemic layers over a symbolic knowledge graph; near-simultaneous with this draft and the most conceptually adjacent prior work. Both target reasoning domains outside SE.

The hypothesis graph sits at the intersection of three lineages: cognitive-architecture memory (Soar / ACT-R / EPIC), LLM-agent memory systems (CoALA / AriGraph / Mem0 / Zep), and typed-belief representations (CausaLab / BeliefMem / Theorem-of-Thought / CMM). The hypothesis graph adopts the Soar memory typology directly as its slot vocabulary, adding only the specific content of the `smem` slot: Peirce-typed, kill-conditioned, designed for LLM prose read/write. Adjacent work: Kirk, Wray & Laird 2023 (AAAI), an LLM-port of the Soar lineage; CoALA (Sumers et al. 2023/24, arXiv:2309.02427); AriGraph (Anokhin et al. 2024/25, arXiv:2407.04363), the closest precedent for graph-structured LLM-agent memory; CausaLab (Yang et al. 2026, arXiv:2605.26029); BeliefMem (Liao et al. 2026, arXiv:2605.05583), strong adjacent on uncertain alternatives with mechanical update.

CMM (Khalid & Arora 2026, OpenReview; a day before this draft) is the closest comparison: the same persistent typed DAG of reasoning artifacts, but observe-and-consume (it types a trajectory post hoc and graduates skills) where ours is perturb-and-falsify (kills fire live, the graph routes the run). The directions are opposite and complementary, the ~385 committed graphs in `sweep/repo-hypotheses/` exactly the corpus its graduation pipeline could consolidate.

Four 2026 systems each carry one component this work combines; what is new here is the join; each piece already exists in one of them. FVDebug (arXiv:2510.15906) builds an actual hypothesis graph for debugging, with a frontier and accumulated evidence, but selects the next node by asking the model, the arbiter this work removes. From Hypotheses to Factors (arXiv:2604.26747) runs the same perturb-and-falsify loop, falsifiable hypotheses behind a deterministic engine over an append-only trace, locked to quantitative finance where this work claims the general semantic-memory substrate. Portable Agent Memory (arXiv:2605.11032) is the nearest provenance memory, a Merkle-DAG that cites the Soar lineage and makes every node reconstructible by content-addressing, but it certifies integrity (the recorded bytes are untampered) where the replay invariant here certifies warrant (the node still survives its trial). And the provenance survey From Agent Traces to Trust (arXiv:2606.04990) enumerates exactly the relations this work mechanizes, Support, Contradict, Invalidate, and names "how provenance quality should be evaluated" as an open problem; the

hypothesis graph is one answer, with replay as the quality bar and the kill condition as an executable edge rather than a descriptive label.

What the mechanism experiment (§7) adds to this comparison is which piece is decisive: not the graph structure, which several of these share, but where a kill edge gets its ground truth. FVDebug lets the model arbitrate the next step; the contrast that separates a narrow fix from a general one is whether the kill is graded against the model's own belief or against an external oracle, and the externalized form is released as `abductor` (enumerate a case space, calibrate against a known-good reference, gate). A model can rebuild the enumeration for itself but not that oracle (§7.8), which is the axis these neighbors leave implicit.

A second cluster treats truth and uncertainty as first-class rather than a downstream score: NARS, OpenCog's AtomSpace/PLN, Nanopublications (Groth et al. 2010), and, closest in time, Traxia (arXiv:2606.08256), converging on these primitives two days after Truth Is Buildable (2026-06-04). Where each stops short of a replayable, kill-conditioned entitlement ledger is adjudicated in Verifiable Knowledge, the paper that owns the epistemology. What is specific here is the data structure: none makes that meaning the semantic contract of a memory node, truth operationalized by replayable edge structure rather than a stored label or textual provenance record.

> Production LLM memory systems with graph variants (Zep/Graphiti, Mem0), staged-hypothesis selection in science agents, deterministic gating in adjacent settings, and reflective memory systems (Reflexion, DebugMate) are surveyed in the appendix; they are adjacent on particular axes but do not change the comparison spine.

9.4 Adversarial filtering and termination

System	Domain	Stage operated at	Visibility regime	Cross-family
Multi-Agent Debate (Liang et al. 2023/24, arXiv:2305.19118)	General reasoning	Patch / answer stage	Open (cross-visibility)	Single model family
Refute-or-Promote (Agarwal 2026, arXiv:2604.19049)	Defect discovery	Review stage	Asymmetric context	Yes
This work	SE (industrial code)	Pre-patch hypothesis stage	Blind challenge (no cross-visibility)	Yes (Sonnet + GPT-5.5)

Table 2. Adversarial multi-model filtering: this work occupies the pre-patch / blind cell. Termination disciplines (λ_A's type-theoretic proofs, SafetyDrift's absorbing states) sit at composition or trajectory scope where this work's verdict-routed gate sits per-instance.

Closest in spirit is POPPER (arXiv:2502.09858), which runs agentic sequential hypothesis tests under e-value error control, the same sequential-testing machinery this project's `inquire` workflow uses to classify evidence. POPPER terminates statistically, on an error-rate bound over a population of tests; this work terminates mechanically, on a deterministic kill predicate over a single binary verdict, and persists the outcome as replayable memory where POPPER's tests are ephemeral.

10 Limitations

The mechanism evidence is existence-grade. One audited divergence, on one instance, in a program that was not preregistered when it ran. The pilots' nulls are confounded by a selection artifact we can name but not yet remove (the triage fast-path, §8.1), and the localization-hard band where the mechanism should live has not been decisively tested; its one strong candidate did not reproduce at HEAD. Nothing here is a rate.

The smem is small and per-instance. Hypothesis graphs in this work are one markdown file per inquiry; cross-instance accumulation is untested. That carries its own deflationary point: the file was never the bottleneck at any repo size, so heavier stores need to earn their keep at cross-instance scale, where the per-instance case never demanded them.

How to refute this. The central claims are built to fail loudly, each against a committed artifact a hostile auditor runs rather than a promise.

- The mechanism claim dies if the Verus receipt fails to replay, or if a discriminating program shows the externally-oracled fix wrong where the maintainer's general fix is right. The clean, forced-fresh

dataset and both patches are committed to check, and the flux trail at `flux-1613-trail-v1` is there for the same test on the auditability case.
- The encoding-boundary claim, that the oracle is not self-generable, dies if a self-graded model reaches the hard side with labels it authored independently of the predicate under test. That would show the oracle is inducible after all, and closing the gate's coverage on that side is the next experiment's preregistered target.
- The methodology claim dies if, as audited cases accumulate, an external oracle stops separating from self-grading everywhere a receipt can see. The oracle's source is then not the active ingredient, and the mechanism README names that null as this thesis's own falsifier.
- The attribution dies if the oracle bracket fails to replicate from its preregistered sample.
- The novelty claim, a capability lift driven by a domain-general reasoning tool, dies if the lift is shown to ride on domain-specific tooling rather than the general operation, or if a documented prior result already demonstrated the same from a tool that carries no answer and no domain model (§9.2). The comparative search (§14) publishes the queries to re-run.

Generator staleness. The checkpoints are fixed (Sonnet 4.6 era), and a more capable generator narrows the band where the mechanism is observable, since it resolves more cases unaided. The mechanism claim therefore survives only in the regime where verification is the bottleneck, which is also the regime the discussion argues matters.

11 Future work

The program reorganizes around the smem, in order of leverage.

- The preregistered mechanism hunt. The Verus and flux cases define the target band: localization-hard, verification-bottlenecked bugs where the symptom sits far from the cause and the suite cannot see the difference between fixes. Each loop now carries its deductive step before it runs: testing X, predict Y, refuted by Z (`METHODOLOGY-preregistration.md`, `CANDIDATES-localization-hard.md`). Three to five audited existence cases, or the committed null, is the next paper.
- Coverage as the design lever, the oracle from approved history. The Verus dissection (§7.7, §7.8) names two handles the next experiments turn directly. Gate coverage sets the generalization frontier, so widening the enumeration to the genuine-divergence-preserve shape is a falsifiable test of whether the model then carves out the XOR's hard side. And the oracle a model cannot induce is mined from human-approved history, the merged fix, the regression suite, the resolved-issue label, so calibrating `abductor` differentially against base and the approved fix is the concrete way to close a gate's blind spot without authoring a fresh oracle by hand. Prospectively, on a bug with no merged fix yet, the hard side has no correct verdict until a human spends the judgment.
- Conjecture: abduction is the necessary, and necessarily external, operation of inquiry. Read the dissection one level up. Every warrant-producing step contracts the symmetric difference between what the inquirer believes and what is true, locating one case the current hypothesis mishandles and removing it. Generation cannot find that case, because adding hypotheses has no floor that says this one is missing; only a disagreement evaluated against a reference outside the inquirer's belief can, which is why the externalized oracle was the active ingredient. If that is right, the oracle's source is the operation inquiry is made of. The same contraction recurs across the companion posts (refutation in What Cannot Be False Cannot Be True, stranger-replay in Verifiable Knowledge, the fold in Compress and Unfold), which is suggestive rather than evidential and is kept here as a conjecture. We do not lean on it as a citation. It dies if a recorded trace reaches the same held-out general predicate from self-authored labels only, with no external label, replay, human-approved reference, or environmental execution entering before the discriminating step; lucky guesses (no warrant) and already-solved cases (the difference already empty) do not count.
- Conjecture: in-context abduction degrades with diff size. Hold the harness fixed and the claim is about the operands: a model's ability to compute the XOR in its own context should fall as the diff and case-space grow: a large structured operation attempted in prose rather than executed. Two regimes, one of them a conjecture. The endpoint needs no experiment: once the operands exceed the context window, in-context computation is impossible, so externalization is the only option at scale, and a codebase outgrows any finite window eventually. The interior, within the window but large, is the falsifiable part. Computing the XOR is a synthesis task, relating cases to one another to find the one the hypothesis mishandles, so the pairs to weigh grow with the square of the case-space and the load rises quadratically where a fact-lookup rises linearly; in-context accuracy should degrade as the operands grow, and faster than linearly. The underlying fall-off is the context-rot phenomenon

(Chroma 2025); we have no direct measurement of XOR accuracy against diff size, and flag that rather than borrow a result that grades something else. It dies if a within-harness sweep over diff and case-space size shows in-context XOR accuracy flat or rising with size. The Verus receipt is a single point on that axis.

- The hypothesis graph as a type. Specify the abstract data structure independently of this harness: operations (perturb-and-classify, generate-edge-from-kill, prune, replay), the soundness invariant (every node reconstructible from its recorded trial), and a serialization any agent can emit. The goal is an interchange format for auditable reasoning, so that "show your work" becomes a machine-checkable demand rather than a rhetorical one.
- Grade the trace, beyond the patch. Bench design follows from the audit: report determinacy-aware denominators; build instruments whose oracle is expensive or absent and whose causes are hidden, because that is where method separates from reach; and score submissions on replayability, so a suite-green over-narrow fix (§7) stops being indistinguishable from a root-cause one.
- Receipts-first contribution as the deployment lane. The flux submission is the template: fix, discriminating receipt, soundness twin, full suite, replayable graph, residual flagged to the experts. Agent PRs are drowning in justified slop suspicion; the trace is the antidote, because it converts "trust my patch" into "audit my ledger." Scaling that lane (and measuring maintainer response to trace-backed PRs) tests attestation-displacing-trust ecologically. The first maintainer merge of a trace-backed fix on a maintainer-stuck issue is the lane's golden ticket: an adversarial expert accepting the work on its ledger, with the doer invisible to the verdict.
- Cross-instance smem accumulation. Let the graph grow across instances within a repo, then across repos within a domain; the current work tests the smem only at per-instance scope.
- Concurrent diagnosis over a shared graph. The monotone graph (nodes append, kills idempotent) permits a lock-free fan-out: parallel agents on one shared graph re-verifying each other's kills instead of trusting them, with transitive accountability the precondition that keeps it safe, a conflict-free blackboard for the inquiry layered over git's blackboard for the code. Running it, and measuring the wall-clock win, is the to-do.

12 Conclusion

In the domain where every step is checkable, an agent's reasoning can be made accountable rather than merely trusted. The hypothesis graph is a replayable record that submits each generated step to a world-facing trial and retains only what survives. The path from diagnosis to merge becomes a sequence of falsifiable commitments a third party can rerun rather than a verdict it must take on trust. This reduces the cost of relying on an agent from reproducing its work to rerunning its record. The guarantee holds only where the work is perturbable, where a trial can be run and read; extending that region is left open, and the method claims nothing beyond it.

Inside that checkable domain sits a bolder result. On a single contamination-free bug, the externalized comparator carried Sonnet 4.6 to a fix Fable could not reach without it: a capability lift of two tiers or more, where a scaffold usually buys only automation. To our knowledge it is the first public demonstration of a lift that wide. A controlled ablation pins it to the comparator alone, encoded reasoning in the harness, no training involved.

This work is independently funded by the author. Every claim here ties to a committed receipt, the nulls included.

13 Availability and reproducibility

- Repositories. github.com/kimjune01/swebench-pro (the bench run; frozen tags `prereg-pro-v1`, `prereg-pro-v1-cheap`), github.com/kimjune01/swebench-verified (prior-generation baseline, Zenodo-DOI'd), github.com/kimjune01/swebench-pro-audit (the determinacy audit; every claim one row in `CLAIMS.md`, all 728 verdicts in `COVERAGE.md`, mechanical spine re-derivable by grep), github.com/kimjune01/determinacy (the audit as a portable tool for any SWE-bench-shaped bench; SWE-rebench run included), github.com/kimjune01/hygraph-mechanism (the mechanism experiment; the Verus #2219 lift with its clean, forced-fresh dataset and the 43,586-line climb trace, and the flux trail frozen at `flux-1613-trail-v1`; the README's artifact index maps every claim here to its committed path), github.com/kimjune01/abductor (the externalized kill condition as a standalone, domain-general instrument: enumerate, calibrate against a known-good baseline, gate, with the `/debug` skill that drives the loop).

- Provenance artifacts. Per-instance trajectories, hypothesis graphs, captured diffs, gate traces, and cost ledger under `runs/scored/artifacts/`; preregistrations at the freeze SHAs; the OSS ledger (`pr-receipts.jsonl`) with the GraphQL query that recomputes every number it asserts.
- Models and disclosure. No training, fine-tuning, or learned weights in the harness; each stage drives a vendor's shipped agentic CLI over the same typed contracts. The bench run paired a Sonnet 4.6 generator (extended thinking on) with a GPT-5.5 challenger (reasoning off, a point against reading the gate as model deliberation); the open-weight pair-swap ported the result wholesale, and each scored instance reads only its own artifacts. Versions and billing mode are in the cost ledger and the freeze SHAs.
- OSS deployment trace. ~385 hypothesis graphs at `kimjune01/sweep/repo-hypotheses/`, one per investigated issue; PR-level outcomes pinned at `kimjune01/kimjune01@paper-2026-05-28`.
- Replication. Boxes, budget, the per-instance cost ledger (`COST_BASIS.md`), and the step-by-step rerun live in the run repo and the field guide How Not to Run SWE-bench Pro.
- Companion writing. The instrument story and field guide: How Not to Run SWE-bench Pro. The error corrected here, from the inside: Precisely Wrong. The epistemology the discussion rests on: Verifiable Knowledge (DOI) and its frame What Cannot Be False Cannot Be True; its buildable-truth core in brief: Truth Is Buildable. Dated provenance posts establish parallel rather than derivative development: Theory is load-bearing (2026-03-17), The proof manual (2026-04-05), and Type the question (2026-04-08) predate ADI (2026-04-17); Evidence has a trajectory (2026-04-27) and The Hypothesis Graph (2026-04-28) predate CMM (2026-05-26).
- PDF. Arxiv-shape build at june.kim/assets/the-hypothesis-graph-semantic-memory-methodeutics.pdf, rebuilt from this markdown source by md2arxiv; the source is canonical.
- DOI. Prior-generation verified artifact: Zenodo-DOI'd. Mechanism experiment (hygraph-mechanism): 10.5281/zenodo.20691973. Pro bundle (swebench-pro): 10.5281/zenodo.20691977. This paper: DOI pending.
- License. Skills released under CC-BY-SA-NS (june.kim/cc-by-sa-ns); repo-level terms in each `LICENSE.md`. The harness an outsider clones is the same harness that produced the published numbers.

Reproducibility invitation. Nullius in verba. Every number here is recomputable from committed artifacts: the bench verdicts by re-running the official grader on captured diffs, the audit's mechanical spine by grep, the flux divergence by replaying the receipt programs against both committed patches. Doubts should be filed as issues against the relevant repository; confirmed corrections fold into the next versioned artifact, as the retraction noted in §7 and the reversal this version reports already have.

LLM collaboration disclosure

LLMs enter this work in three roles. Subject of study: the harness under evaluation uses frontier LLMs as generator and challenger, with versions, billing mode, and provenance disclosed in §13 and the artifacts. Instrument: model pairs adversarially verify the audit's two-expert tier (one constructs, an independent family refutes) and blind-judge the mechanism pilots, always with the mechanical layer (grep, grader, replay) holding the verdict. Writing aid: the prose was drafted and revised with Anthropic's Claude (Opus 4.8 and Fable 5) from human-authored outlines and session notes, with adversarial review from OpenAI's codex (GPT-5.5); the claims, methodology, numbers, and argument structure are the author's. No LLM decided what to publish.

Acknowledgments

We thank John Laird for endorsing this submission, and the flux maintainers for engaging with a stranger's receipts on their hardest open issue.

14 Novelty and comparative search protocol

- Why this section exists. Several claims here take the form "we found no prior X." Such claims are only as honest as the search they rest on. This section publishes the queries so an auditor can re-run the search and either confirm the gap or find what we missed.
- Sources searched. Google Scholar; arXiv (cs.LG, cs.AI, cs.CL, cs.SE); ACL Anthology; OpenReview; GitHub code and repository search; public SWE-bench leaderboard and submission archives.
- Queries by claim.

  - Peircean SE-agent loop. "Peirce" "LLM agent" abduction deduction induction software engineering; "abduction deduction induction" "LLM agent" "software engineering"; "Peirce" "SWE-bench" agent.
  - Hypothesis-graph agent memory. "hypothesis graph" "LLM agent" memory; "belief graph" "LLM agent" memory hypothesis; "knowledge graph" "LLM agent" "hypothesis" "memory".
  - Blind multi-model hypothesis-stage filtering. "multi-agent" "code" "hypothesis" "SWE-bench"; "blind" "multi-agent" "code review" LLM.
  - Trajectory-shape termination gates. "LLM agent" termination criteria trajectory; "finite state" "LLM agent" "termination".
  - Benchmark determinacy / construct-validity audits. "SWE-bench Pro" "construct validity"; "underdetermined" benchmark "problem statement" test; "specification" "ambiguity" "SWE-bench" audit.
  - Full per-instance provenance on SWE-bench Pro. SWE-bench Pro leaderboard submissions trajectories cost ledger; site:github.com "swe-bench-pro" "trajs".
  - Sub-$1k Pro replication and per-instance cost ledger. "SWE-bench Pro" "cost per instance"; "SWE-rebench" cost per problem.

- Caveats. The search is best-effort and bounded by visible-web indexing; private industry work and non-indexed venues are not covered. Discoveries of overlapping prior work should be reported as issues for citation update.

14.1 Comparative search supporting the artifact claim

The artifact claim (§13), no method documented publishes per-instance receipts at this depth on SWE-bench Pro, requires a comparative search. The claim concerns receipts, with rate set aside. The bar: published per-instance trajectories, captured diffs, gate or evaluator traces, cost ledger, and reproducible run conditions.

Candidate audit (against the receipt bar). Each top public submission or comparable report is checked for: published per-instance trajectories (T), captured diffs (D), evaluator/gate traces (G), per-instance cost ledger (C), reproducible frozen artifact (R), and resolve rate at or above ours on the same bench. Receipt-bar columns are present (✓), partial (◐), or absent (✗).

Submission / report	Bench	T	D	G	C	R	Rate ≥ ours	Notes
Official `swebench/experiments` repo (multiple top entries)	Verified	✓	✓	✗	✗	◐	Various	Minimum publication norm: trajs/logs/patch.diff/report. No gate traces, no cost ledger.
Top vendor leaderboard entries (Claude Code, OpenHands, SWE-agent, AutoCodeRover)	Verified	◐	◐	✗	✗	✗	Reported below 97%	Submissions report numbers; reproducible bundles and cost ledgers rarely published.
SWE-bench Pro official page (Scale)	Pro	◐	◐	✗	✗	✗	N/A (curator)	Uncapped cost (250-turn limit). No per-instance cost ledger.
Nilenso Pro trajectory analysis	Pro	◐	✗	✗	◐	✗	N/A (third-party)	Cost/token/time analysis across four frontier models. Not a submission.
SWE-rebench public reports	rebench	◐	◐	✗	✓	◐	Below ours	Strong cost transparency (Cursor Composer 2.5 at $0.23/problem).
This work: Verified	Verified	✓	✓	✓	✓	✓	426 / 438 eligible (97.3%)	Companion repo `swebench-verified`; Zenodo DOI; gate traces and cost ledger committed.

Submission / report	Bench	T	D	G	C	R	Rate ≥ ours	Notes
This work: Pro	Pro	✓	✓	✓	✓	✓	694/728 = 95.3%; open-weight pair 678/728 = 93.1%	Same frozen harness, whole eligible set, 0 incomplete; two model pairs under one bundle. Public-split, gate-oracle regime: an artifact claim, not a leaderboard claim (§7).

Reading. No row above the two rows here combines all five receipt-bar columns (T/D/G/C/R) on the same bench. The claim is about receipt depth alone, leaving resolve rate out of it, and survives as long as the table reads this way; a citation showing a fuller combined receipt is the cleanest refutation.

15 Extended intellectual lineage

Foundational sources grounding §4, §3, §2, and §9, collected here so Related Work stays focused on contemporary systems.

15.1 Peircean inquiry and the philosophy of science

- Peirce 1878 (Illustrations of the Logic of Science): the original three-mode taxonomy.
- Peirce 1903 (Pragmatism as the Logic of Abduction): abduction as the only mode that introduces new content.
- Bacon 1620 (Novum Organum); Popper 1934 (The Logic of Scientific Discovery): falsification as the inductive-side constraint.
- Ramsey 1926 (Truth and Probability): operational credence as betting odds; the hypothesis graph's node-level semantics descends from this work.
- James 1907; Dewey 1929: the pragmatist commitment that truth is inseparable from action.
- Meehl 1967; Feynman 1974 ("Cargo Cult Science"): the difference between rigor-shaped activity and actual rigor, the standard §7 holds itself to.
- Kimball 1957; Tukey: the Type III error, the exact answer to the wrong question; the failure mode the determinacy audit (§7) is built to prevent, examined in the companion post Precisely Wrong.

15.2 The hypothesis graph's structural ancestors

- de Kleer 1986 (assumption-based truth-maintenance systems): persistent dependency structures over beliefs with mechanical retraction; the TMS keeps consistency where the hypothesis graph keeps trials, and a TMS justification is not replayable by a stranger.
- Dung 1995 (abstract argumentation frameworks): attack relations between claims as first-class structure; the hypothesis graph's kill edges are attack edges bound to executions rather than arguments.
- Wald 1947 (sequential testing) and Vovk & Wang 2021 (e-values): the sequential-evidence framing that shaped `inquire`'s diagnostic stance. No accumulator is deployed: the code under test is deterministic, so the gate routes on the binary grader verdict (§6.3).
- Pearl 1988; 2000/2009: DAGs as the substrate for structured belief and causal-structure inference; this work borrows the typed-node/typed-edge form and leaves the probabilistic semantics behind.
- Zeller & Hildebrandt 2002 (delta debugging): the canonical demonstration that mechanical perturbation of code is a productive inference primitive; optimization-shape where the hypothesis graph is methodology-shape.
- Reiter 1987 (A theory of diagnosis from first principles) and de Kleer & Williams 1987 (the General Diagnostic Engine): model-based diagnosis as conflict sets, minimal hitting-set diagnoses, and entropy-minimizing choice of the next measurement. The hypothesis graph runs this abductive loop but over an open, LLM-generated hypothesis space rather than a fixed component model, and its "where to perturb next" is the GDE measurement-selection problem handed to the reasoner. Reiter's hitting sets are the theory of how multiple kills jointly localize a cause.
- Clarke et al. 2000 (counterexample-guided abstraction refinement, CEGAR), Solar-Lezama et al. 2006 (counterexample-guided inductive synthesis, CEGIS), and Cousot & Cousot 1977 (abstract interpretation): the closest formal kin to the hypothesis graph's loop, in which a counterexample is a kill that names the next refinement. The hypothesis graph generalizes that loop from a closed abstraction domain or synthesis grammar to an open hypothesis space, which forfeits the completeness and sound-by-construction refinement that closure buys (the open-world relevance boundary is

undecidable in general, reducing to Rice's theorem) and recovers soundness only per node, through replay. Two pieces of their machinery are unported and natural to inherit: the real-versus-spurious counterexample check (a principled test of whether a kill is genuine or an artifact) and widening (a principled stop that over-approximates with stated precision loss, the formal form of declaring an inquiry's trajectory oscillatory and committing to the general shape).

15.3 Bi-abductive and compositional inference

- Calcagno et al. 2009: compositional shape analysis via bi-abduction; Facebook Infer as the industrial-scale instance of typed-mode inference on real code.
- O'Hearn 2019: separation logic and incorrectness logic.
- Zilberstein, Saliling & Silva 2024 (arXiv:2305.04842): Outcome Separation Logic; tri-abduction for branch composition.
- Bylander et al. 1991: abductive computational complexity.