The Methodeutic Harness on SWE-bench Pro

Hypothesis graphs as agent semantic memory, grounded in Peircean methodeutics

June Kim

2026-05-28

Contents

Download PDF — arxiv-shape preprint, rebuilt from this source.

0.1 Abstract

Reasoning can be encoded at the harness layer. This paper is the existence proof: a *methodeutic harness* (methodeutic is Peirce's term for the methodology of inquiry) running a typed, re-entrant inquiry over a persistent hypothesis-graph memory, closed by a deterministic gate that no model arbitrates. Frontier models propose, critique, and implement within this structure, but the harness owns the state, the evidence record, and the decision to continue, branch, commit, or stop.

Each software-engineering instance is treated as a problem of inquiry. Recon performs abduction by writing competing hypothesis nodes from the observed failure. Craft performs deduction by tracing consequence-edges from a selected hypothesis into an intervention. Audit performs induction by reading back tests, diffs, logs, and reviewer evidence, then witnessing or killing leaves in the graph. The gate consumes the resulting trajectory shape and routes the next move. Termination is a property of the typed evidence trace, not a model self-judgment. Adversarial filtering operates at the hypothesis stage, not the patch stage: blind-blind pushout runs two frontier models on the same evidence pack with no cross-visibility; the merge surfaces disagreements as the next investigation edge.

The system assembles existing lineages rather than introducing a new model: Peircean inquiry modes, bi-abductive program analysis, sequential evidence accumulation, invariant-feature splitting, and Voyager-style observe-hypothesize-test-commit loops. The contribution is the runnable assembly: a fully automatable agent harness for industrial code that persists hypotheses, exposes its reasoning trace, and separates generation from arbitration.

The harness is validated on three surfaces that grade different failure modes. On SWE-bench Verified, it resolves 426 of 438 eligible instances under the official grader, with public per-instance trajectories, captured diffs, gate traces, and a cost ledger. Under the same frozen harness, a preregistered SWE-bench Pro run resolves 694 of 728 eligible instances (95.33%), the whole eligible set graded in one measurement, 0 incomplete, with the same provenance committed per instance. In the wild, the same pipeline has produced 80 agent-selected, agent-authored, agent-submitted PRs merged across 46 open-source repositories under issue-first sampling: 53% merge rate under adversarial maintainer grading, GraphQL-verifiable. In corpus form, ~380 committed hypothesis graphs expose the triage and investigation traces that preceded deployment decisions.

The audited per-instance cost on the Pro run is ~$3 (Sonnet API-mode ledger), with the GPT-5.5 challenger on a Max subscription at marginal zero. A pure-API replicator projects ~$2.2k for the Pro run on the Sonnet side, plus the challenger-side API rate, still inside an individual researcher's discretionary spend.

The evidence standard is receipt-level: every claim ties to a trajectory, a captured diff, a gate trace, and an audited cost line. The existence proof (that a methodeutic harness with hypothesis-graph memory and a deterministic gate resolves industrial software-engineering instances under the official SWE-bench grader) is discharged by the Verified component (426 / 438 eligible, public, Zenodo-DOI'd). The Pro component is preregistered, frozen by git tag, and now terminal at 694 / 728 eligible (95.33%); it extends the witness to a contamination-resistant tier. Conditional on §3.10 provenance publication, the *comparative* claim asserts that no documented method known to us has shown a higher official-harness resolve rate at lower audited per-instance dollar cost, reproducibly across both SWE-bench Verified and Pro under one frozen harness with equivalent receipts.

No model training, fine-tuning, reinforcement learning, curated dataset, or GPU resources are required. The result is an *organization of inference*: typed inquiry, hypothesis-graph memory, adversarial model disagreement at the hypothesis stage, and deterministic arbitration outside the model.

0.2 1. Introduction

Agents acting on engineered systems have nowhere to put what they currently believe is going on. Prose context is lossy; skill libraries store verified code, not falsifiable claims; vector retrieval indexes documents, not hypotheses.

Existing graph-memory work for agents builds adjacent infrastructure without imposing typed reasoning-mode contracts on read and write; adjacent and parallel systems are surveyed in §9.2a–b. The belief substrate, in the systems we are aware of, remains undertyped.

This paper presents a persistent typed hypothesis graph for LLM-agent semantic memory. Peircean reasoning modes are first-class node types. Falsification criteria are executable edge predicates. Pipeline stages have mode-specific write contracts. Routing and gating are performed by deterministic graph operations rather than model calls.

The specific composition is tri-disciplinary by construction: it synthesizes Peircean epistemology (1878, 1903), the cognitive-architecture memory tradition (Soar; Laird 1987; ACT-R; chunking-style mechanical operations on structured memory), and LLM-agent read/write semantics. Graph memory for agents is old; Soar has had it since 1987. We borrow Soar's memory typology (semantic memory / `smem`, procedural memory / `pmem`, episodic memory / `epmem`) only as vocabulary. The hypothesis graph occupies the `smem` slot, the pipeline-stage skills occupy the `pmem` slot, the per-run trajectories occupy the `epmem` slot. The fuller mapping is in §9.2b. Our search (§8) did not find a prior `smem` instance that combines LLM-shaped prose read/write, Peirce-typed falsifiable nodes, Vovk-Wang e-value kill-conditioned edges, one-mode-per-stage write contracts, and model-free deterministic gate operations.

The methodeutic harness is the typed read/write contract over that smem. Each stage is constrained to one Peircean mode (1878, 1903): recon writes only abductive nodes, craft reads abductive nodes and emits deductive consequence-edges, and audit reads predictions and writes inductive verdicts plus a trajectory classification. Stages cannot freelance into each other's mode. The Peircean vocabulary names the contract surface; the pipeline enforces the contract mechanically, not the philosophy.

The headline claim is narrow and falsifiable. The Verified half is discharged now; the comparative claim across both benches becomes fully unconditional once the Pro artifact is DOI-pinned (§3.10). No method documented

in our comparative literature search (§8), known to us, has *demonstrated* with equivalent receipts a higher SWE-bench official-harness resolve rate at a lower audited per-instance dollar cost, reproducibly on the two most popular SWE benchmarks (Verified and Pro) under one frozen harness, than the skill set this paper presents.

The four receipts grade different failure modes:

- *Verified resolve rate* (426 / 438 eligible, Zenodo-DOI'd, trajectories and diffs committed): controlled-bench accuracy.
- *Pro* under preregistration (terminal: 694/728 eligible = 95.33%, 0 incomplete, whole eligible set graded): contamination-resistant accuracy.
- *OSS PR merge rate* (80 merged across 46 repos, 53%, GraphQL-verifiable): ecological maintainer grading.
- *~$3/instance Sonnet-API (codex on Max subscription, marginal $0); ~$2.2k Sonnet-side for the Pro run, ledger published*: accessibility.

The load-bearing properties are *demonstrated* (we publish the trail that makes the numbers auditable) and *reproducibility on both benches* under one artifact. Refutation is a citation showing a stronger combined receipt.

Adversarial filtering operates at the hypothesis stage. Blind-blind pushout runs two frontier models from different families with no cross-visibility, then merges on disagreement while the worktree is still untouched. Agreement is uninformative; disagreement supplies the next investigation edge. Termination is mechanical: a deterministic gate consumes the audit-classified evidence trajectory (Wald 1947, Vovk & Wang 2021) and routes re-entry, completion, or budget exhaustion. The gate is finite-state over trajectory shape, attempt count, budget, and frontier-closure status. No model sits in the gate.

SWE-bench is the legibility surface. Two tiers run under one frozen harness: Verified (saturated, contaminated, ports cleanly as a baseline) and Pro (contamination-resistant, the primary validation surface). Same frozen loop, two contamination regimes, two independent provenance trails. The bench supplies legibility; the methodeutics is the contribution.

The contribution is the composition. The primitives (Peirce's three modes, Soar's memory typology, Ramsey's credence under stakes, Pearl's typed directed graph, Wald and Vovk-Wang's evidence-trajectory vocabulary, Calcagno's bi-abduction, Voyager's loop shape) are honored in §2 as the lineages the design descends from. The runnable assembly is what the paper exhibits; the harness executes on industrial code under contamination discipline, or it doesn't.

Part of what the assembly exhibits is knowing what the score does not show. The harness marks its unexercised affordances (cyclic graphs, the flaky-test reading of non-clean trajectories), keeps its claims inside what the bench can witness, and reports the boundary rather than the headline. The resolve rate is the legibility surface's output; the calibrated, checkable method is the artifact.

0.3 2. Theoretical grounding: methodeutics as a contract surface

Peirce's *Illustrations of the Logic of Science* (1878) and *Pragmatism as the Logic of Abduction* (1903) type the operations of inquiry into three modes that are not interchangeable. 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?*

The modes are typed by what they cannot do. Abduction proposes content but does not test it; induction tests but introduces no new explanatory content; deduction traces consequences but invents nothing. Collapsing them produces familiar failure modes: confirmation bias (induction without abductive alternatives), confabulation (abduction without inductive grounding), free-association (no typed mode at all). Modern LLM agents collapse the three by default; a single forward pass proposes, predicts, evaluates, and rationalizes in undifferentiated prose.

Methodeutics is Peirce's term for the methodology of inquiry, the systematic study of how to conduct the typed-mode loop well. The framework draws on a dispersed primary-source lineage that this paper cites directly rather than through any single secondary work.

Modes of reason and the irreducible three. Peirce 1878, 1903; Bacon 1620 (induction); Popper 1934 (falsification); Meehl 1967 (the "soft science" critique); Feynman 1974 (cargo-cult science).

Pragmatist credence and stakes-indexed belief. Ramsey 1926 (*Truth and Probability*; subjective probability, the Dutch Book argument, belief as betting odds); James 1907 (*Pragmatism*; truth as what works); Dewey 1929 (*The Quest for Certainty*; truth as warranted assertibility). The node-level semantics of the hypothesis graph descends from this lineage.

Bi-abduction, tri-abduction, and compositional inference. Bylander et al. 1991 (abductive complexity); Calcagno et al. 2009 and O'Hearn 2019 (bi-abductive shape analysis; the anti-frame/frame decomposition for sequential composition, scaled industrially in Facebook Infer); Noam Zilberstein, Saliling & Silva 2024 (arXiv:2305.04842, OOPSLA), *Outcome Separation Logic*, which extends the bi-abductive lineage to branching composition under nondeterministic and probabilistic effects (their Theorem 5.1: soundness of tri-abduction for reconciling two branch preconditions). We use this as a branch-composition analogue, not as a completeness result for arbitrary abduction.

Evidence trajectories and anytime-valid inference. de Finetti 1937 (subjective probability); Ville 1939 (martingales); Wald 1947 (sequential testing); Robbins 1967; Chernoff 1959; Kelly 1996 (capital-growth ties to e-values); Shafer 2021; Grünwald 2024; Ramdas 2023.

Shape vocabulary. Milnor 1985; Strogatz 2014: the source of the four words (convergent / divergent / oscillatory / chaotic), borrowed as vocabulary; the dynamical-systems treatment is separate work, not relied on here.

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). The data structure we use (typed nodes with directed edges) is Pearl's lineage applied to hypothesis representation rather than to causal-structure inference, with one structural departure: we do not require acyclicity. Pearl's DAG is acyclic by definition; acyclicity is what licenses d-separation and the conditional-independence factorization. Inquiry on engineered systems can concern cyclically-causal domains (cascading failure, retry storms in SRE-style debugging), so the structure must admit cycles to stay faithful. Pearl is therefore the ancestor we depart from structurally, not only the skeleton we inherit.

Working instantiations of the loop in different fields. Calcagno et al. 2009 (Facebook Infer, bi-abductive shape analysis at industrial scale); Arjovsky et al. 2019 (Invariant Risk Minimization, tri-abductive figure-ground splitting under environment variation); Wang et al. 2023 (Voyager, observe?hypothesize?test?commit in Minecraft with a skill library, untyped at the node level). Adjacent and parallel work (IDEA, ADI, CMM, AriGraph, CausaLab, CoALA, and others) is treated in §9.

Composition over the hypothesis graph. Three of the lineages above enter the design through separate roles. The structural skeleton is Pearl (1988, 2000): the DAG is Pearl's representation primitive for structured belief. We borrow the typed-node/typed-edge form but relax acyclicity, not the semantics (probabilistic conditional independence, do-calculus). Nodes here are typed hypotheses; edges encode dependence and the kill conditions that fire on evidence. Because the structure no longer guarantees acyclicity, termination rests entirely on the deterministic gate (budget, attempt count, frontier closure; §3.5), not on graph topology. The node semantics is credence under stakes (Ramsey 1926; James 1907; Dewey 1929). Each hypothesis-graph node carries a *belief* at a credence level, stakes-indexed and continuous, not a fact. Confident confabulation, high apparent confidence over a node that has not earned it, is the named failure mode that the Peircean stage-typing and the kill-condition discipline are jointly designed to prevent. The reasoning-mode label types *what kind* of belief the node carries; the credence types *how much*. The update semantics is a qualitative trajectory-shape label borrowed as vocabulary from the sequential-evidence lineage (Wald 1947; Vovk & Wang 2021; Ramdas 2023), with the four-word shape vocabulary echoing dynamical systems (treated in separate work). Across audit re-entries the evidence pattern is labeled convergent / divergent / oscillatory / chaotic (§3.4), and the label routes the gate. We borrow the vocabulary only; we deploy no numerical e-value accumulator (the IID Bernoulli assumption SWE-bench audits violate would be required) and make no claim that this framing is epistemically superior to a binary verdict. Pearl + Ramsey + sequential-evidence vocabulary: three primitives, three roles, integrated in one data structure.

0.4 3. Method

Figure 1. The methodeutic harness used on SWE-bench Pro. Stages are typed by Peircean mode (recon = abduction, craft = deduction, audit = induction); the gate is finite-state with no model call; the outer loop re-enters recon with an updated graph rather than retrying the patch.

0.4.1 3.1 The inquiry frame

Each SWE-bench instance is recast as an inquiry on an engineered system: a failure trace, a codebase, a question of root cause, and an intervention that must not regress the rest of the system.

Code is the right substrate for this data structure because it combines three properties that other inquiry domains rarely combine. It is *reproducible* (same input yields same output, modulo controlled nondeterminism), *deterministic* (causal lines from input to behavior are mechanical), and *perturbable* (single-line and single-function diffs are cheap to apply and fully observable). Hypotheses about code can therefore be tested by cheap mechanical perturbations and falsified by deterministic predicates; kill conditions are not approximations, they are executions.

One instance is enough to settle the predicate in this regime. Statistics is the machinery for inferring causal relationships from noisy populations; in code the per-instance response is mechanically observable, so a single passing test on a captured diff is a complete verdict that the diff satisfies the executable benchmark predicate for that instance. The verdict is for the predicate, not for the broader notion of root-cause correctness: hidden behaviors not covered by the executable predicate are out of scope, and we report only what the predicate checks, which is what the grader checks. Medical hypotheses need populations because individual responses are noisy; the per-predicate per-instance check here does not. The paper therefore reports counts, denominators, and per-repo breakdowns rather than confidence intervals or significance tests: per-predicate verdicts are already exact, and aggregating them is bookkeeping. Aggregate claims across repos, benches, or run-order remain empirical and are caveated where they appear. Portability of this regime to substrates where the per-instance response is not mechanically observable is open.

The three Peircean modes become three stages with disjoint contracts and disjoint access patterns over the hypothesis graph. Abduction (recon) proposes candidate causes and writes hypothesis nodes with falsifiable predicates and kill conditions; no edits, no external access. Deduction (craft) traces each surviving hypothesis's consequences into a concrete patch; an adversarial challenger critiques the diff against the spec. Induction (audit) runs the test suite, takes the grader's pass/fail verdict, and, across re-entries, labels the shape of the evidence trajectory to route the gate's next move. The verdict stays binary; the shape label is the routing signal layered on top of it.

0.4.2 3.2 Hypothesis graph as recon output

Recon emits a hypothesis graph document, not a patch sketch. The structure is a three-pillar synthesis. The *skeleton* (Pearl 1988, 2000) is a directed graph: nodes are typed hypotheses, edges carry dependence and kill predicates. We adopt Pearl's typed-node/typed-edge form but relax his acyclicity constraint, since inquiry can concern cyclically-causal systems (cascading failure, retry storms); termination is then bounded by the gate, not by topology (§3.5). Every hypothesis graph committed in this artifact is in fact acyclic, no bench instance surfaced cyclic causation, so the cyclic affordance is design-justified but empirically unexercised here (§6). The *node semantics* is credence under stakes: each node carries a stakes-indexed credence in its claim; the Peircean reasoning-mode label types what kind of belief, the credence types how much. The *update semantics* is qualitative trajectory-shape classification (Wald 1947; Vovk & Wang 2021; Ramdas 2023; Milnor 1985): across audit re-entries the evidence pattern is labeled convergent / divergent / oscillatory / chaotic, and the shape label fires the kill condition. §3.4 details the four shapes and why a deployed numerical e-value accumulator is set aside in v1 (IID violation).

Kill conditions are mechanical predicates over the evidence trajectory, not model preferences. A node dies when its predicate fires. The graph persists across iterations; re-entry adds nodes rather than overwriting. The frontier closes when every leaf is killed or witnessed.

Worked example. Drawn from `kimjune01/sweep/repo-hypotheses/antonmedv__fx__413.md` (one of the ~380 publicly committed graphs from the OSS deployment surface), abridged. The issue: under the `snap`-installed build of `fx`, the yank-to-clipboard keystroke flashes the dialog but the clipboard stays empty. An

early recon pass converged on a documentation-only verdict; a later pass re-entered the graph and surfaced a shippable code fix the earlier passes had not generated. The graph (compressed):

```
H0: snap strict confinement blocks fx's clipboard subprocess
  Perturbation surface: main.go:645–660 (yank handler), snap/snapcraft.yaml
  Falsifiable predicate: clipboard.WriteAll() errors under strict confinement
    because the bundled snap cannot exec host xclip / wl-copy binaries; the
    handler discards the error with `_ =`.
  Reasoning mode: deduction.  Credence: 0.95.
  Status: WITNESSED (root cause confirmed against code + snapcraft + maintainer
    thread).

H1: switch snapcraft confinement: strict → classic
  Status: KILLED. Snap Store manual review required; not code-only.
H2: grant desktop / x11 / wayland plugs
  Status: KILLED. plugs grant socket access but do not bundle the host
    binaries the upstream clipboard library shells out to.
H3: bundle xclip / wl-clipboard inside the snap
  Status: KILLED. out of scope per maintainer thread.
H4..H6: README warning / fx.wtf docs / native Go clipboard backend
  Status: KILLED on venue, repo boundary, and library-swap scope respectively.

[Re-entry: prior pass froze at "docs PR" verdict, but the frontier was not
 closed: every leaf had been killed by scope or venue rather than evidence
 against the underlying mechanism.  Audit routed back to recon.]

H7: OSC 52 escape sequence as additive write path
  Perturbation surface: copyToClipboard() wrapper; transitive dep
    github.com/aymanbagabas/go-osc52/v2 promoted to direct.
  Falsifiable predicate: terminal-emulator OSC 52 sequence
    (\x1b]52;c;<base64>\a) emitted to os.Stderr (fx's TUI render channel)
    sets the system clipboard with no external binary and no snap plug.
  Reasoning mode: abduction (option) → deduction (stderr channel, write-only
    use) → induction (build passes; sequence verified).  Credence: 0.80.
  Status: WITNESSED. `go build ./...` passes; sequence correct.  Runtime
    confirmation in a confined terminal pending.
```

Three structural elements are visible in the node-set: typed reasoning-mode labels (`deduction` on H0; the abduction-then-deduction-then-induction trace on H7), stakes-indexed credence (0.95 on the root cause, 0.80 on the proposed fix pending runtime confirmation), and mechanical kill predicates (H1 by Snap Store policy; H2 by the missing-binary deduction; H3–H6 by scope and venue). The H7 discovery on re-entry is the loop's intended behavior: the earlier verdict was structurally premature because H1–H6 closed by scope rather than by evidence against the underlying mechanism, leaving the frontier open. The deterministic gate routed the trajectory back to recon, which proposed H7, and audit witnessed it under a build check. The corpus of ~380 such files is the deployment trace referenced in §5.6 and §7.

0.4.3 3.3 Blind-blind pushout at the hypothesis stage

Two frontier models from different families receive the same evidence pack with no cross-visibility, and each produces a hypothesis independently. A third pass extracts the disagreements, not the agreements: the disagreement becomes the next node in the graph; the agreement is recorded but not actionable. Adversarial filtering operates at hypothesis time, while the worktree is still untouched, rather than at patch time where the diff is already written. Sampling stochasticity alone produces real divergence even within a single model; cross-family divergence compounds it with architectural and training-corpus differences. Both are signal.

0.4.4 3.4 Trajectory-shape classification

Each audit re-entry produces an outcome trace for the active hypotheses: which $FAIL_TO_PASS$ tests flipped, which $PASS_TO_PASS$ tests regressed, which related kill predicates fired. The evaluator labels the shape of that trace across re-entries with one of four qualitative classes. The four words echo dynamical-systems vocabulary; the dynamical-systems treatment is developed in separate work and is not relied on here. We make no epistemological claim in this paper: not that a four-way trajectory label is superior to a binary verdict, and not that the e-value / sequential-evidence paradigm is preferable to fixed-threshold testing. The per-instance verdict is still the grader's pass/fail (§3.1); the shape label is a pragmatic routing input that tells the gate what to do when a single pass hasn't yet been reached across re-entries.

- Convergent. Evidence trends toward witnessing one hypothesis across re-entries (the test responses progressively favor it; sibling hypotheses fade).
- Divergent. Evidence trends toward two or more sibling hypotheses simultaneously (the responses partition the test set rather than concentrating on one).
- Oscillatory. Evidence flips between hypotheses across re-entries without stabilizing (the same node alternates witness and kill across passes).
- Chaotic. No stable trend across re-entries.

In practice the two load-bearing labels are convergent and divergent: these are the clean readings the gate routes on. Oscillatory and chaotic are the non-clean residue, and they have a programmer-native name: a flaky test. Evidence that flips between hypotheses without stabilizing (oscillatory) or shows no trend at all (chaotic) is the signature of an unreliable oracle: a test whose verdict is nondeterministic, so no amount of search converges on it. In principle, then, these shapes have two causes (the harness failing to find the fix, or the test itself being flaky), and the shape label does not by itself distinguish them. Flaky tests are a well-documented reality in production codebases; the phenomenon is real out in the wild. This curated bench has none by construction (SWE-bench oracles are deterministic), so every oscillatory or chaotic instance here traces to harness search-failure, not oracle nondeterminism. The flaky-test reading of these shapes is therefore real but unexercised in this set: the bench excludes the regime, not the world. No terminal Pro or Verified verdict depends on the oscillatory/chaotic interpretation; the headline rates are the grader's binary verdicts, and the shape labels only routed re-entry. Either way they route to re-entry or to budget-exhausted termination, and we read them as a degenerate signal rather than as a validated dynamical regime of the evidence. The four labels are exhaustive (every trace falls into one), but we impose no further structure on them: no ordering, no metric, no claim that the partition carries more inferential weight than the grader's verdict. We report the labels because the gate consumes them.

We do not deploy a numerical accumulator: there is no per-test-outcome multiplicative likelihood-ratio score with a fixed threshold in v1, and a formal anytime-valid accumulator would require IID Bernoulli outcomes that SWE-bench audits do not satisfy. The qualitative labels are what the gate consumes; a numerical accumulator is left as future specification, with no claim that it would resolve more instances than the labels do.

The deterministic gate (§3.5) reads the shape label emitted by the evaluator. Both the evaluator and the gate are mechanical; neither is a model call.

Broad-strokes labeler (canonical implementation in `driver/shape.py`):

```
def classify(trace):
    # trace: list of audit re-entries, each carrying the
    # active-hypothesis -> outcome map for FAIL_TO_PASS and
    # PASS_TO_PASS plus fired kill predicates.
    winners = [dominant_hypothesis(t) for t in trace]
    if len(set(winners[-3:])) == 1 and witness_strength_increasing(trace):
        return "convergent"
    if multiple_hypotheses_partition_tests(trace[-1]):
        return "divergent"
    if winners and winners != sorted(winners) and any_node_flipped_witness_kill(trace):
        return "oscillatory"
    return "chaotic"
```

`dominant_hypothesis`, `witness_strength_increasing`, `multiple_hypotheses_partition_tests`, and `any_node_flipped_witness_kill` are pure functions over the captured trace; no model in the loop.

0.4.5 3.5 Deterministic gating

The gate is a finite-state classifier over (trajectory shape, attempt count, budget remaining, frontier-closure status), not a model call. Outputs are `continue-into-craft`, `re-enter-recon-with-graph-update`, `terminate-success`, `terminate-budget-exhausted`, and `escalate-to-human` (private set only). No stage may decide its own termination; the gate decides. The gate's logic is published, reviewable, and frozen by the same tag as the rest of the artifact. Termination is mechanical, which is what makes the run reproducible: re-running the same evidence trace through the same gate yields the same routing.

The gate's transition table (canonical implementation in `driver/gate.py`):

```
def gate(shape, attempt, budget_left, frontier_closed):
    if budget_left <= 0:
        return "terminate-budget-exhausted"
    if frontier_closed and shape == "convergent":
        return "terminate-success"
    if shape == "convergent":
        return "continue-into-craft"
    if shape in ("divergent", "oscillatory"):
        return "re-enter-recon-with-graph-update"
    if shape == "chaotic" and attempt >= MAX_OUTER:
        return "escalate-to-human"      # private set only
    return "re-enter-recon-with-graph-update"
```

Five inputs, five outputs, no calls into the model. The same `(shape, attempt, budget_left, frontier_closed)` tuple maps to the same routing on replay.

0.4.6 3.6 Outer-loop iteration

Audit failures do not retry the patch. They route back to recon with the trajectory classification and the updated graph. Craft sees a different node-set on re-entry rather than the same node twice. The attempt budget per instance is bounded. Budget exhaustion counts as a clean termination without convergence, and the loop treats it as a verdict. No peek at held-out tests at any point in the loop. The gate's inputs are local to the instance and the agent's own run.

0.4.7 3.7 Fault classification (operator-side)

Pre-registered fault classes cover environment-induced failures: auth outage, quota exhaustion, infra timeouts. These are platform faults, distinct from reasoning losses. Classification triggers on fault-window membership, not on instance verdict. Wins inside the window get re-run alongside losses. Any reclassification rule that lets a losing run be retried while letting a winning run stand is disallowed; asymmetry is exactly the loss-laundering the protocol forbids.

0.4.8 3.8 One-shot held-out discipline

The public set may be iterated through the outer loop and re-run on platform fault. The held-out set is graded once on a frozen artifact, with no per-instance feedback consumed as an iteration signal; the held-out grade is treated as an oracle, never a stopping signal. The artifact that earns the submission hash is the submission. Local-green/official-red is structurally impossible because the hash binds the captured prediction to the run that earned it.

0.4.9 3.9 Preregistration and freeze

The artifact is frozen by an annotated git tag (`prereg-pro-v1`); every scored-run artifact cites the freeze SHA. A new tag opens a new worklog with the failure class that justified the restart named explicitly. v1 is the first scored tag.

0.4.10 3.10 Provenance contract

Per-instance trajectories (agent sessions for each pipeline stage) are captured off-box on a polling cadence. A hypothesis graph document, captured diff, grader output, and gate trace are committed per instance, alongside a per-box ledger and cost provenance. A run earns headline status only when full provenance is published; scores alone are insufficient.

0.5 4. Experimental Setup

0.5.1 4.1 Datasets and eligibility

Two benches run under one frozen loop. SWE-bench Verified is a curated subset, saturated (top public submissions resolve >85% as of mid-2026) and training-cutoff-contaminated for current frontier models; we use it as a baseline that the loop ports cleanly, with companion repo `swebench-verified` and a frozen artifact under Zenodo DOI. SWE-bench Pro is the live contamination-resistant tier with a public/held-out split across different repos, and it is the primary validation surface. Both runs use the official harness; no bespoke graders. Defect lists are documented per bench (`KNOWN_BAD.md`) and cover bench defects only, never our failures; the eligible set is the full set minus documented defects.

The two-bench design dissociates bench saturation (Verified) from loop generality (Pro). Same loop, two contamination regimes, two independent provenance trails. If the loop carries from one to the other at comparable per-repo rates, the assembly is not bench-specific.

0.5.2 4.2 Execution

Fixed run order, no early stopping. Whole-set evaluation on a multi-box fleet with per-instance isolation. Dynamic coordination with fault-tolerant resume across box reprovisioning.

0.5.3 4.3 Models and roles

No training, no fine-tuning, no reinforcement learning, no in-context demonstrations from prior instances. Frontier models are queried via API at their published checkpoints; the harness contains no learned weights of its own. Generator and challenger are drawn from current frontier families, deliberately cross-family to avoid mode collapse on a single training prior. Models are pluggable: the methodeutic harness and smem operate over any pair of capable frontier models with structured-output and tool-use support. We freeze the specific model versions used for this run's reproducibility, but no part of the methodology is specific to a particular model. Auth source is not part of the frozen artifact; billing mechanism is orthogonal to evaluation discipline.

Prior-instance information boundary. For each scored bench instance, the loop reads only artifacts derived from that instance's failing-test reproduction and its own in-cycle writes; no other instance's hypothesis graphs, trajectories, or solutions enter the model context. Pattern reuse exists at the harness-design level (typed stages, kill-condition vocabulary, gate transition table), not at the per-instance prompt level.

0.5.4 4.4 Grading

Official harness only; no bespoke graders. Bench defects are noted in one line per instance; no per-instance forensic analysis that risks becoming bench-specific tuning.

0.5.5 4.5a Cheap-model ablation (in flight, pre-publication)

Before publication, the same frozen harness is rerun on Pro only with Cursor Composer 2.5 as generator and Gemini Flash 3.5 as challenger, replacing the Sonnet + GPT-5.5 cross-family pair. Both are recent cheap models, and the cross-family adversarial property in §3.3 is preserved (Cursor × Google). Pro is the contamination-resistant tier and is where the ablation is most informative; Verified is contaminated for every submitter and ports cleanly under any reasonable model pair. SWE-rebench publishes Composer 2.5 at ~$0.23/problem; Flash 3.5 adds a near-zero increment on the challenger pass; combined per-instance cost projects under $0.50, or ~$290 to grade the full Pro eligible set (728 instances).

Three outcome readings frame it. If the result holds at a comparable rate, the typed contracts and the kill-conditioned graph are doing the work and model selection is not the lever; this closes the "model selection is unablated hyperparameter" critique. If the result degrades modestly, the loop is necessary but not sufficient, and frontier capability matters on the hard-tail repos where the cheap pair falls short. If the result collapses, frontier capability is the dominant factor and the loop's contribution is negligible. The answer is published regardless. The Composer+Flash run carries the same provenance contract as the headline run (per-instance trajectories, captured diffs, gate traces, cost ledger) and ships under the same Zenodo DOI as the Sonnet+codex artifact: one bundle, two model-pair runs on Pro.

As of 2026-05-30 this run is in flight, hovering near 90% on the partial set graded so far. If the terminal result stays near this level it would fall in the "holds at a comparable rate" band for a ~7× cheaper, fully cross-family pair, but the figure is provisional pending termination and full provenance, and is reported here to mark the run's status, not as a result. The final number and per-repo table land at termination on the same publication discipline as the headline run.

This is a robustness run, not a clean isolation. The model-pair swap changes model family, capability, training cutoff, and cost-per-instance simultaneously; cross-family adversarial filtering is held fixed, but the families themselves change (Anthropic+OpenAI → Cursor+Google). It tests whether the harness still produces results under a ~7× cheaper pair (combined ~$0.40/instance against the Sonnet-API rate of ~$3/instance in this work; codex side ran on Max subscription, not API). Single-factor attribution is out of reach here. Clean per-factor ablations (typed-mode contract on/off, blind-blind pushout on/off, gate determinism on/off) remain future work and are scoped in §10. Both Composer 2.5 and Flash 3.5 are recent enough that their training cutoffs differ from the headline pair's; comparable performance across two distinct family axes is informal evidence the methodology is not riding on the specific pretraining of any one model or family.

0.5.6 4.5 Cost and replication envelope

Two replication modes are validated in this work, and this run uses both in parallel. *Subscription mode*: a single frontier-vendor consumer subscription (~$200/month) covers the run at a multi-week pace, gated by per-window quota. Marginal cost at the boundary is zero; time cost is weeks. Used for the codex challenger throughout this run, and for the Sonnet generator in the early window before the API switch. *API mode*: pay-as-you-go API access at the published rate, validated Sonnet-side cost approximately $3 per instance blended across light and heavy repos; ~$2.2k Sonnet-side projection for the Pro run. The two modes compose: the recorded ledger here is Sonnet API + codex subscription; a pure-API replicator pays Sonnet-side at this rate plus the vendor's published rate for the codex side. Faster, capacity-bounded by the fleet's box count rather than the quota window.

Compute requirements outside the model are minimal. The harness runs on commodity Linux boxes (we used 4× small EC2 instances during the API window): no GPU, no curated training data, no offline pipeline. Per-instance dollars and wall-time are committed to the ledger alongside the verdict and the trajectory, so replicators can audit what the run cost in real currency.

A consumer subscription or a ~$3k API budget puts replication within reach of graduate students, independent researchers, and small teams; no special funding required.

0.6 5. Reported metrics

> Because the official grader returns deterministic per-instance verdicts (§3.1), the paper reports counts, denominators, and per-repo breakdowns rather than fitting aggregate significance tests. Per-instance verdicts are exact for the executable predicate; aggregation is bookkeeping. Cross-bench and cross-repo comparisons are presented as empirical claims, not inferential statistics, and are caveated where they appear.

0.6.1 5.1 What the tables contain (per-bench)

Counts of WIN, LOSS, and INCOMPLETE for each bench, with denominator (eligible set = full set minus documented `KNOWN_BAD.md` defects at the freeze SHA). Per-repo rows record repo, instance count, WIN, LOSS, INCOMPLETE, mean cost per instance, and median wall-time. Aggregating across repos is a weighted average over uneven coverage; we report the per-repo table as the honest read and let aggregation be a derived computation the reader can do. The cost ledger commits per-instance dollars and wall-time alongside the trajectory; replicators can audit the total run cost down to the instance. Hypothesis-graph metrics per terminal instance include graph depth at termination, frontier-closure status (closed or budget-exhausted), count of kill-conditions fired, and count of nodes added across all outer-loop iterations. The gate-routing distribution reports the frequency of each gate output (`continue-into-craft`, `re-enter-recon`, `terminate-success`, `terminate-budget-exhausted`); the deterministic gate's behavior is itself a reportable property.

0.6.2 5.2 Treatment of incomplete and faulted instances

Fault classes (§3.7) are `AUTH_OUTAGE`, `QUOTA_EXHAUSTED`, `INFRA_TIMEOUT`, `CRAFT_HANG`. Classification triggers on temporal-window membership regardless of instance verdict; wins inside the window get re-run alongside losses, and asymmetric re-runs are forbidden. Each faulted instance gets one re-run; terminal verdicts stand, and faulted-again rows remain `INCOMPLETE` in the final ledger. Out-of-window verdicts stand: no reclassification of a real-output verdict on grounds of post-hoc inspection. `INCOMPLETE` gets its own column alongside `WIN` and `LOSS` rather than being folded into a denominator-mangling adjustment, so the reader sees the bounded honest cost of the run.

0.6.3 5.3 Two-bench reading

For each repo present in both eligible sets, we report the per-repo Verified count and Pro count with denominators. Both numbers per repo are presented side by side, and whether the assembly carries is left to the reader's interpretation. We do not report Wilson CIs, paired tests, or equivalence procedures. The per-instance verdicts are deterministic; aggregation is bookkeeping. A reader who wants confidence intervals can compute them from the published counts.

0.6.4 5.4 Independent re-grade

Twenty random WINs from the union of both benches, stratified by repo to avoid light-repo bias, drawn with a fixed random seed committed before the re-grade. The re-grade protocol applies the captured diff plus the official harness on a clean container, by an independent operator who has not seen the trajectory; it confirms or refutes the original WIN verdict per instance. The output is a `k / 20` confirmation count plus per-instance notes on the (20−k) non-confirmations if any. The confirmation count is compared against the published rate.

0.6.5 5.5 Composer-2.5 robustness run (§4.5a)

Same counts, same tables, computed independently for the Composer-2.5-only run. The Sonnet+codex per-instance verdict and the Composer-2.5 per-instance verdict are reported side by side for each instance in the intersection. The disagreement count is read directly; no statistical procedure applied.

0.6.6 5.6 Current results

Verified (closed, public). 426 / 438 eligible WIN under the official grader. Denominator-explicit; per-repo table committed to the `swebench-verified` repo's `results/` directory; Zenodo DOI pins the frozen artifact. The full 500-instance Sankey (eligible 438 = 500 − 44 sphinx-doc offline-infeasible − 18 documented `KNOWN_BAD.md` defects) is in the companion README.

Pro (terminal, frozen tag `prereg-pro-v1`). - 694 WIN / 34 LOSS = 95.33% on 728 eligible (731 dataset instances − 3 gold-patch defects excluded pre-run and documented). 0 INCOMPLETE: the whole eligible set carries a terminal verdict, graded in one measurement. - Run span ~3.5 days (first dispatch 2026-05-27, last verdict 2026-05-30), through three provider-credential stalls and a mid-run Max-subscription ? paid-API billing switch, with 0 instances lost. - Per-repo (11 public repos): ten of eleven resolve at 92.3% or above; NodeBB is the lone outlier at 74.4% and contributes 11 of the 34 losses. Per-repo W/L, %win, and runtime quantiles are committed. - No development-overfit signal. The harness was iterated on Verified (all Python) before this run; if that overfit the pipeline, the development language should resolve *higher*. It resolves lower: Python (dev language) 94.7% versus non-Python (Go/TS/JS, never touched in development) 95.7%. The strongest language (Go, 98.6%) and the weakest (JS/NodeBB, 74.4%) are both novel. The freeze does not defend against overfit here; this direct check does. - All 34 losses have non-empty captured patches: every loss is the loop producing a fix the official tests rejected, not failing to produce one. A by-hand read of the diffs (a provisional characterization, not a re-grade) puts the true *capability*-loss count below 34: at least one is a verified serialization defect (capture stripped the diff's string-literal quotes, making it unparseable, a manufactured false loss), ~4 are capture/serialization defects, and the leading band (~19) is gate-vs-official-mismatch *candidates*, where the local audit was satisfied but the official grader rejected, where a stricter audit gate, not necessarily a smarter model, might close the gap, though some may be genuine capability misses until a re-grade confirms. These hand-read categories are a provisional split, not a measured one, and are not yet claimed to exhaust the 34. - Independent re-grade spot-check: no binding leak. Six cross-language WINs, re-graded from the committed diffs on fresh containers under the unmodified official grader with no agent re-run, all six reproduced RESOLVED (6/6).

The per-instance Pro artifact bundle (trajectories, captured diffs, gate traces, and cost ledger for all 728) is committed at the frozen tag; the Zenodo DOI pinning it is forthcoming, on the same publication discipline as the Verified artifact (§3.10).

The provisional 2026-05-28 snapshot read 97.0% on the 402 early-order instances; the final number is lower, as that snapshot's own projection anticipated: the heavy tail (NodeBB and the harder repos) sat in the then-ungraded remainder, and grading it pulled the rate down to 95.33%.

Cost (Pro ledger). Run is hybrid: Sonnet generator on Max subscription throughout (marginal cost zero) except for a brief May 27–28 API-mode window opened when subscription quota was exhausted and closed when quota replenished; codex challenger on Max subscription throughout (marginal cost zero). Sonnet-API observed rate across the closed window: ~$3 per audited instance (per-instance ledger committed alongside trajectories). Subscription-mode portions are marginal-zero and don't appear in the ledger. The full bench at the observed Sonnet-side API rate projects ~$2.2k for the generator side if run pure-API end-to-end; a pure-API replicator adds the challenger side at the vendor's published rate.

Verification cost. Verification doesn't require re-running the full bench. A representative re-grade sample (the §5.4 protocol uses N = 20 stratified-random WINs) costs ~$3 × N at the observed Sonnet-API rate: ~$60 for the planned re-grade. The bar to audit this work is API access, an AWS account (or equivalent compute), and discretionary spend in the tens of dollars.

OSS deployment trace. 80 PRs merged across 46 repositories at the time of writing under adversarial maintainer grading; 53% merge rate (merged / all agent-submitted PRs under the `sweep` campaign tag). 67% positive maintainer engagement on 65 issues filed since the slop-filter campaign start (engagement = reproduce-confirmed, assigned, or commented as actionable). All denominators and definitions GraphQL-verifiable from the pinned queries at `kimjune01/kimjune01@paper-2026-05-28/README.md`. The profile README is volatile; the receipt is the pinned commit. ~380 hypothesis graphs publicly committed in `sweep/repo-hypotheses/` are the upstream funnel from which submissions are drawn.

Operator role in the deployment trace. No human-authored code in any of the 80 merged PRs, no prompt-level

steering during the runs, and no human curation of which issues to work on. The harness samples *issues* (not repos) at random from a universe defined as filed issues on ?200-star open OSS repos across roughly 9 languages, then runs recon / craft / audit autonomously, writes the patch and PR body, and submits. The full deployment story (slop-filter campaign, drip discipline, maintainer-pushback evolution) is in the companion post Speedrunning Open Source. Issue-first sampling matters: every attempted instance is a problem someone filed because they wanted it fixed, which sidesteps the failure mode of repo-first sampling (pick a codebase, contrive work for it). The human role is strictly upstream (define the harness, define the sampling universe) and operational (kick off runs, manage cost), not authorial, not directive, not selective. The 53% merge rate is the rate at which adversarial maintainers accept *agent-selected, agent-authored, agent-submitted* code into codebases the operator does not maintain, on problems the operator did not pick.

0.7 6. Limitations

Public Pro is training-cutoff-contaminated for both model families used. This is a property of the bench, shared by all submitters, not an isolation of method as the cause. Self-holdout is weaker than Scale's: different commits, same repos. Cross-repo generalization to Pro's held-out partition is untested locally. Training-cutoff math for both models is published explicitly, with per-instance date overlaps disclosed.

One-shot held-out discipline is verifiable only at submission time. The submitter-side invariant (no held-out feedback in the artifact) is necessary but not sufficient. The run is cost-bounded; heavy repos may be underrepresented or over-budgeted depending on order, with cost and time logged per instance.

Selection of generator and challenger families is a hyperparameter we do not ablate. The ablation that would isolate the loop's effect (with vs without the typed-mode constraint, on one fixed model) is a separate clean-room study. Because no training or fine-tuning occurs, pretraining-cutoff contamination is the channel most directly auditable from the artifact (cutoffs vs instance dates). Other channels (leaderboard feedback, harness tuning, known-bad-list leakage, retry policy, run-order effects) are handled by §3.7–§3.10 (preregistration, frozen harness, one-shot held-out, published failure list, frozen run order).

The smem is small. Hypothesis graphs in this work are per-instance; cross-instance memory accumulation is future work. The smem proposal is therefore tested in its simplest non-trivial regime.

Cyclic graphs are an unexercised affordance. The structure permits cycles by design (§3.2): it relaxes Pearl's acyclicity so it can stay faithful to cyclically-causal domains such as SRE cascading failure. No instance in this corpus surfaced cyclic causation (every one of the committed hypothesis graphs is acyclic, checkable against the public artifact), so the cyclic path is design-justified but empirically unexercised here. The claim is a scope fact about the corpus, not a doubt about the design: the affordance is built, and the bench simply did not enter the regime that would exercise it.

Causal attribution. This paper is a strong artifact demonstration; clean isolation of the loop's contribution from the model's contribution rests on the §4.5a Composer+Flash robustness run, now in flight (early returns near 90%, provisional). Until that run terminates and the clean per-factor ablations in §10 are done, the attribution is artifact-level, not factor-level: the body shows the harness clears the bench, not that the harness rather than the models is what clears it. The in-flight cheap-pair signal is suggestive of the former but not yet conclusive.

0.8 7. Discussion

> Both surfaces are closed. Verified: 426 / 438 eligible, frozen, Zenodo-DOI'd, public. Pro: 694 / 728 eligible = 95.33%, 0 incomplete, whole eligible set graded under the official grader at frozen tag `prereg-pro-v1`. Pro landed 1.9 points below Verified (97.26%); the per-repo and loss readings below say where the gap lives.

The harness ports cleanly to industrial code across both contamination regimes. 97% of eligible Verified instances and 95.33% of eligible Pro instances resolved under the official grader with full per-instance provenance: captured diffs, trajectories, and cost ledger committed. Verified is public and Zenodo-DOI'd; the Pro bundle is committed at the frozen tag with its DOI forthcoming. The hypothesis-graph smem records typed inquiry decisions per run (kill conditions fired, evidence trajectories classified); whether those records prove reusable across instances

and across benches remains future work at cross-instance scope. The deterministic gate produces reviewable termination traces: an external auditor can replay its decision against the captured trajectory and budget, and the routing is reconstructible from the evidence trace alone.

Reproducibility comes from the captured trace, the frozen artifact, the deterministic gate, and the qualitative shape labels feeding it, not from Peirce. Peirce names the contract vocabulary and makes it legible; the mechanical enforcement (stage contracts that reject mode-freelancing inputs, captured trajectories with hypothesis-graph snapshots, finite-state gate logic, deterministic shape-to-routing mapping) is what lets the run replay.

The composition was designed for principled reasons. The hypothesis graph's shape (Peirce-typed nodes, e-value-inspired kill-conditioned edges, one-mode-per-stage write contracts, model-free gates) was designed to close known failure modes of LLM agents: mode collapse, confabulation, unauditable revision, vibes-based termination. The design rationale is independently inspectable from the artifact, and Verified shows the design clears industrial code at a 97.26% eligible rate. Pro replicates it within 1.9 points (95.33%), across a repo set with zero overlap with the development surface, and the gap is concentrated, not regime-wide. Nine of eleven Pro repos resolve above 95%; the deficit lives almost entirely in NodeBB (74.4%). The development-overlap check is the load-bearing one: the development language (Python) resolves *lower* than the never-developed languages (Go/TS/JS), so the Verified?Pro gap is not contamination-regime asymmetry leaking through; it tracks per-repo difficulty. At the per-repo level the assembly reads as bench-agnostic and contamination-regime-agnostic, with NodeBB the one repo that names where the harness is weakest.

The OSS PR record (80 merged across 46 repos, 53% merge rate, adversarial maintainer graders; GraphQL verifiers pinned at `kimjune01/kimjune01@paper-2026-05-28/README.md`, since the profile README itself drifts) reports what the harness does on contact with non-curated codebases. The receipt is *agent-selected, agent-authored, agent-submitted* (§5.6, and the longer story in Speedrunning Open Source): the harness samples issues at random from a 200-star-floor universe across ~9 languages and runs the full loop without human keystrokes in the diff or natural-language steering during operation; the human role is harness-design and operations. This closes the four reviewer reflexes that dismiss agent-coding demos (cherry-picked issues, fabricated problems, vibe-coded patches, developer-assist authorship) at once. The ~380 publicly committed hypothesis graphs at `kimjune01/sweep/repo-hypotheses/` are the deployment trace: skip verdicts, dead-end paths, engage decisions. This is ecological; the deployment surface lacks a within-instance comparator, and isolating the typed-mode constraint requires the in-flight Composer+Flash robustness run (§4.5a). The merge rate is well above the AI-slop floor visible in the same pinned commit's slop table for contemporaneous unstructured attempts.

Portability follows from the no-training stance. The harness contains no learned weights of its own; anyone with frontier-model API access can clone the repo, swap in their model pair, and run the loop without training compute, curated datasets, or GPU resources. Weights are pluggable; the cost envelope (§4.5) fits inside an individual researcher's discretionary spend.

One peripheral note on bench infrastructure: held-out validity is partly a function of who gets graded. Published access rubrics (the way submission formats are published) would strengthen Pro's claim to be community infrastructure. We offer this as a structural observation.

One further structural observation, on token efficiency. The ~$3/instance figure (§5.6) translates to substantially fewer tokens per resolved instance than vendor-published agent runs at comparable resolve rates on Pro-class instances: the per-instance ledger committed alongside the trajectories lets a reader compute the comparison directly at API-published per-token rates. Token efficiency at this scale appears as a harness-level property in this artifact, not a model property; the underlying models are the same frontier checkpoints anyone else calls. The lever is orchestration craft, and the artifact that carries it is a markdown file plus pinned dependencies, distributable at git-clone latency.

One framing note in closing. This work is one chapter of a broader program: a compiler from prose to executable agent behavior, of which the methodeutic harness is the intermediate representation, the hypothesis graph the typed memory, and skills the compiled units. The program is developed at length in the methodeutics textbook; we use *"compiler"* descriptively, in the LLVM (Lattner & Adve 2004) and DSPy (Khattab et al. 2023) lineage of typed pipelines from specification to reproducible behavior. This paper is the SWE-bench workload through that pipeline; the broader claim is developed elsewhere.

In the LLVM / GCC tradition where the compiler is built with itself, elements of this harness were developed by applying earlier versions of the harness to its own design tasks (the longer treatment); the published receipts are from the post-bootstrap frozen artifact.

0.9 8. Search Methodology

- Why this section exists. Several claims in this paper 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 readers 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*; *"Peircean" "LLM agents" abduction deduction induction*; *"abduction deduction induction" "LLM agent" "software engineering"*; *"Peirce" "SWE-bench" agent*; *"code agent" "abduction" "deduction" "induction" LLM*.
  - Hypothesis-graph agent memory. *"hypothesis graph" "LLM agent" memory*; *"belief graph" "LLM agent" memory hypothesis*; *"knowledge graph" "LLM agent" "hypothesis" "memory"*; *"AriGraph" "knowledge graph world models" "episodic memory" "LLM agents"*.
  - Blind multi-model hypothesis-stage filtering. *"multi-agent" "code" "hypothesis" "SWE-bench"*; *"multi-model" "code agent" "disagreement"*; *"blind" "multi-agent" "code review" LLM*; *"ensemble" "LLM agents" "software engineering" "SWE-bench"*.
  - Trajectory-shape termination gates. *"LLM agent" termination criteria trajectory*; *"finite state" "LLM agent" "termination"*; *"sequential testing" "LLM agents" stopping rules*.
  - Full per-instance provenance on SWE-bench Pro. *SWE-bench Pro leaderboard submissions trajectories cost ledger*; *"SWE-bench Pro" "trajectory" "cost"*; *"SWE-bench Verified" "trajectories" "cost ledger"*; *site:github.com "swe-bench-pro" "trajs"*.
  - Two-bench validation under one frozen artifact. *"SWE-bench Verified" "SWE-bench Pro" "same" "scaffold"*; *"SWE-bench Pro" "Verified" "frozen" "artifact"*; *"SWE-bench Pro" "SWE-bench Verified" "cross-bench"*.
  - Sub-$1k Pro replication and per-instance cost ledger. *"SWE-bench Pro" "cost per instance"*; *"SWE-bench Pro" "cost ledger"*; *"SWE-bench" "cost-per-instance" "leaderboard"*; *"SWE-rebench" cost per problem*.
- Caveats. The search is best-effort and bounded by visible-web indexing; private industry work, in-preparation manuscripts, and non-indexed venues are not covered. Discoveries of overlapping prior work post-publication should be reported as issues against the repository for citation update.

0.9.1 8.1 Comparative search supporting the headline claim

The headline claim, *no method documented has proved a higher SWE-bench resolve rate at a lower audited per-instance cost with equivalent receipts*, requires a comparative search separate from the novelty search above. The queries below are scoped specifically to that claim. The bar for *equivalent receipts* is: published per-instance trajectories, captured diffs, gate or evaluator traces, cost ledger, and reproducible run conditions.

Sources searched. SWE-bench Verified leaderboard (github.com/swe-bench/experiments); SWE-bench Pro official page (scaleapi.github.io/SWE-bench_Pro-os/); SWE-rebench public reports (swe-rebench.com); Nilenso Pro trajectory analysis; OpenHands, SWE-agent, AutoCodeRover, Aider, Devin reports; venturebeat / techcrunch reporting on Pro/Datacurve / DeepSWE; arXiv recent submissions tagged SWE-bench; vendor blogs (Anthropic, OpenAI, Google) on agent benchmark performance.

Queries. - *"SWE-bench Verified" "per-instance" cost trajectories*; *"SWE-bench Pro" leaderboard "cost" "trajectories"*; *"SWE-bench" submission "cost ledger"*. - *"SWE-bench" "cost per instance" submission diff trajectory*; *"swebench" submission reproducible cost*. - *OSS PR merge rate LLM agent maintainer-graded benchmark*; *agent-produced pull request acceptance rate*. - *"SWE-bench Verified" 95% 96% 97% top resolve rate trajectories cost*; *Pro leaderboard top resolve rate cost*.

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 the rate this paper reports on the same bench (Rate). Receipt-bar columns are *present* (Y), *partial* (~), or *absent* (—). A row that doesn't combine all six is not a refutation of the headline.

Submission / report

Bench

T

D

G

C

R

Rate □ ours

Notes

Official swebench/experiments repo (multiple top entries)

Verified

Y

Y

—

—

~

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.

Datacurve / DeepSWE reporting (VentureBeat)

Pro-class

—

—

—

~

—

N/A (journalism)

Reports e.g. GPT-5.5 ~$5.80/trial median (2× this work's per-instance rate). Journalism-level cost data, short of a structured ledger.

SWE-rebench public reports

rebench

~

~

—

Y

~

Below ours

Strong cost transparency (Cursor Composer 2.5 at $0.23/problem); reported resolve rates below the rates this paper documents on Verified.

This work: Verified

Verified

Y

Y

Y

Y

Y

426 / 438 eligible (97.3%)

Companion repo swebench-verified; Zenodo DOI; gate traces and cost ledger committed.

This work: Pro

Pro

Y

Y

Y

Y

Y

terminal 694/728 = 95.33%, 0 incomplete

Same frozen harness; whole eligible set graded in one measurement. Artifact bundle committed at frozen tag; Zenodo DOI forthcoming (§3.10).

Reading. No row above this paper's two rows combines all six receipt-bar columns and a resolve rate at or above the rates this paper documents on the same bench. The headline survives as long as that table reads this way.

Caveat. Top-line resolve rates above the receipt bar may exist in private submissions or in submissions whose receipt set we have not been able to verify; the headline is bounded by what we documented. A citation showing a stronger combined receipt is the cleanest refutation.

0.10 9. Related Work

0.10.1 9.1 SWE-bench, contamination, and cost transparency

The SWE-bench family defines the Verified / Pro lineage, official harness, and contamination-resistant tier design. SWE-rebench (swe-rebench.com) uses post-cutoff filtering as a parallel contamination strategy and publishes per-problem cost figures (e.g., Cursor Composer 2.5 at $0.23/problem), used here as the price anchor for the model-agnostic robustness run in §4.5a. Adaptive data analysis (Dwork et al.) provides formal grounding for held-out-budget discipline. The official SWE-bench experiments repo (github.com/swe-bench/experiments) requires `trajs/`, `logs/`, `patch.diff`, `report.json`, and `test_output.txt` per submitted instance, establishing the minimum publication norm; our provenance contract extends it with gate traces, hypothesis graphs, and cost ledger. The Nilenso SWE-bench Pro trajectory analysis (nilenso.github.io/swe-bench-pro-cost-token-time-analysis) reports deterministic intent classification, exit statuses, structural markers, and cost/error signals across four frontier models on Pro; closest precedent on Pro cost transparency, short of a per-instance dollar ledger. The Datacurve DeepSWE leaderboard reports median per-trial cost for frontier models on Pro-class instances (e.g., GPT-5.5 at ~$5.80/trial median); journalism-level cost data, short of a structured ledger.

0.10.2 9.2 Agent scaffolds, embodied loops, and SE-agent harnesses

SWE-bench-targeted agent harnesses include OpenHands (Wang et al. 2024), SWE-agent (Yang et al. 2024), and AutoCodeRover (Zhang et al. 2024/25). These are methodological neighbors with different scaffold trade-offs; none implements Peirce-typed stage contracts or a kill-conditioned hypothesis-graph memory. Voyager (Wang et al. 2023) is the closest loop-shape precedent: embodied observe?hypothesize?test?commit in Minecraft. We adopt the loop shape, type its stages with Peircean modes, substitute kill-conditioned hypothesis graph for skill library (falsifiable belief in place of verified code), and re-substrate to industrial code. Invariant Risk Minimization (Arjovsky et al. 2019) is tri-abductive figure-ground splitting under environment variation; a third witness

to the loop pattern from distributional generalization. SWE-Replay (2026) evaluates a test-time scaling method across Verified, Pro, and Multilingual; adjacent on cross-bench evaluation, runs without a frozen-artifact preregistration.

Two concurrent developments arrived independently at adjacent points in the same design space, each carrying one of the two structural components this paper composes. Naming them explicitly is the cleanest way to state the contribution: Theorem-of-Thought (Abdaljalil et al. 2025) is a multi-agent framework that types reasoning into abductive, deductive, and inductive specialist agents per query: the typed-cycle component, run without a persistent typed memory across cycles. Cognitive Memory Manager (Khalid & Arora, ACM CAIS AgentSkills 2026) extracts a typed-node DAG ("reasoning graph") by observing agent execution and mines it for recurring patterns to promote to portable SKILL.md files: the typed-graph component, run without a methodeutic harness driving the writes. Neither work is a precedent in the lineage sense; each is a sibling reaching one of the two halves independently. That three labs converged on these decompositions without coordination is itself the structural evidence: typed reasoning and typed memory are landing as natural primitives in this design space. Provenance for the framing developed here is timestamped on the project blog (see The Hypothesis Graph and Evidence has a trajectory).

What this paper composes: the methodeutic harness writes the typed graph prospectively (with kill conditions ahead of evidence and trajectory-shape labels at audit time), the harness reads the graph to drive the next move, and a deterministic gate closes the cycle on trajectory shape rather than model self-judgment. Same graph primitive as CMM, opposite epistemological direction: CMM's graph is descriptive (mined from traces); the one here is generative (it routes the run). Same Peircean typing as ToT, pinned to persistent nodes that survive across cycles rather than dissolved per query. The composition is what the bench result is testing.

Table 1 lays out the cell-by-cell comparison spine for the systems treated below in §9.2a–§9.2b; the prose adds nuance the table can't carry.

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

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

Peirce, agent-level

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 trajectory shape

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 §9.2a–§9.2b. Adversarial filtering and termination are separately tabled in §9.2c and §9.2d.

0.10.3 9.2a Peirce-typed reasoning for LLM agents (parallel work)

IDEA (He et al. 2025, ACL Findings, arXiv:2408.10455), *Enhancing the Rule Learning Ability of Large Language Model Agent through Induction, Deduction, and Abduction*, explicitly cites Peirce and uses the three modes in an interactive LLM-agent rule-learning benchmark. ADI (Gilda & Gilda 2026, arXiv:2604.15727, April 17 2026), *Structured Abductive-Deductive-Inductive Reasoning for LLMs via Algebraic Invariants*, gives an explicit Peircean tripartite protocol with epistemic layers (L0/L1/L2) over a symbolic knowledge graph; near-simultaneous with this paper's draft and the most conceptually adjacent prior work. Both target reasoning domains outside SE (rule learning, algebraic invariants); this paper targets SE agents on real industrial code under benchmark and adversarial-maintainer evaluation. Table 1 carries the cell-by-cell methodological diff.

0.10.4 9.2b Graph-structured memory and typed-belief representations for LLM agents

The hypothesis-graph assembly 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 / falsifiability representations (CausaLab / BeliefMem / Theorem-of-Thought / CMM). The decades-old Soar memory typology (semantic / procedural / episodic) is the slot vocabulary this paper adopts directly (the hypothesis graph fills the `smem` slot, pipeline-stage skills fill `pmem`, captured trajectories fill `epmem`), adding only the specific content of the `smem` slot (Peirce-typed, kill-conditioned, designed for LLM prose read/write). Adjacent LLM-agent work:

- Kirk, Wray & Laird 2023 (AAAI): Soar/ITL agent queries an LLM and encodes into Soar-shaped memory; LLM-port of the Soar lineage.
- CoALA (Sumers et al. 2023/24, arXiv:2309.02427): cognitive-architecture framework for language agents with modular memory.
- AriGraph (Anokhin et al. 2024/25, arXiv:2407.04363): knowledge-graph world models for LLM agents in TextWorld; closest precedent for graph-structured LLM-agent memory.
- CausaLab (Yang et al. 2026, arXiv:2605.26029): LLM agents maintain evolving structural-causal-model hypotheses in a DSL; strong adjacent on persistent inspectable hypothesis representation.
- BeliefMem (Liao et al. 2026, arXiv:2605.05583): candidate conclusions with non-LLM Noisy-OR updates; strong adjacent on uncertain alternatives + mechanical update.
- Theorem-of-Thought (Abdaljalil et al. 2025, arXiv:2506.07106): abductive / deductive / inductive *agents* produce traces structured into formal reasoning graphs; the strongest Peirce-mode adjacent, but modes operate at the agent/trace level rather than typing persistent-memory nodes.

CMM (Khalid & Arora 2026, *From Observed Reasoning to Stable Skills*, Agent Skills '26, OpenReview; published one day before this draft) is the closest contemporary precedent on typed-DAG coding-agent memory and warrants its own comparison. CMM captures coding-agent execution into a typed DAG (seven node types: HYPOTHESIS / INVESTIGATION / DISCOVERY / PIVOT / DEAD_END / SOLUTION / CONTEXT_LOAD), graduates recurring patterns into stable `SKILL.md` files under a human-approval gate, and serves them to future runs.

Convergent role, categorically different runtimes. Both systems converge on the same role for memory: a persistent, typed, queryable DAG of reasoning artifacts. The runtimes diverge on agency. CMM is observe-and-consume: an external coding agent perturbs, CMM types the trajectory post-hoc, future runs consume the graduated skills. Our loop is perturb-and-falsify: craft applies the patch, audit runs the test, kill conditions fire mechanically during the live inquiry. CMM extracts and serves; our harness *is* the actor. CMM applies types at extraction time at the consumption layer; we apply types at write time at the production layer (each stage mechanically rejects mode-freelancing inputs). Same data structure, opposite epistemological direction.

Cold-start vs prior session history. CMM's demonstrated advantage in its published case study depends on six months of one developer's prior sessions, with graduation thresholds of 3+ sessions per project-scoped pattern. Our loop operates on previously-unseen instances cold, with zero per-developer tuning. Different evaluation regimes; the asymmetry is the contribution of memory format, not memory quality.

Complementary by construction. The ~380 hypothesis graphs publicly committed in `sweep/repo-hypotheses/` are exactly the kind of structured corpus CMM's graduation algorithm could ingest to extract specialized per-repo skills. A future system could chain them: our loop produces structured inquiry traces across many instances; CMM's graduation pipeline consolidates those traces into developer-or-repo-specific skills. Table 1 (above) carries

the cell-by-cell methodological diff on typing locus, persistent structure, edge update, and gate; this prose covers the temporal nuance the table can't.

> *Production LLM memory systems with graph variants (Zep/Graphiti, Mem0), staged-hypothesis selection in science agents (DeepScientist), domain-specific hypothesis swarms (drug discovery), deterministic gating in adjacent settings (MemLineage, ROE Gate, Certifying-Risks, FVA-RAG, GHS-TDA), and reflective memory systems (Reflexion, DebugMate, Potpie) are surveyed in the appendix. They are adjacent in particular axes but do not change the cell-by-cell comparison spine above.*

0.10.5 9.2c Multi-model adversarial filtering

Table 2 compares adversarial multi-model setups on stage of operation, visibility regime, and cross-family use.

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 pushout (no cross-visibility)

Yes (Sonnet + GPT-5.5)

Table 2. Adversarial multi-model filtering: comparison axes are stage and visibility, where this work occupies the pre-patch / blind cell.

0.10.6 9.2d Agent termination and trajectory analysis

Table 3 compares termination disciplines. The axes are *what* the system reads to stop, *how* it stops, and the scope at which the stopping decision lives.

System

Stopping signal

Mechanism

Scope

λ_A: Typed Lambda Calculus for LLM Agent Composition (2026, arXiv:2604.11767)

Bounded fixpoint

Type-theoretic termination proof

Composition-level

SafetyDrift (2026, arXiv:2603.27148)

Absorbing state

Markov-chain risk analysis

Trajectory-level

This work

Evidence-trajectory shape (convergent / divergent / oscillatory / chaotic)

Deterministic finite-state gate over (shape, attempts, budget, frontier closure)

Per-instance

Table 3. Agent-termination disciplines. The shape-label gate this paper deploys sits at the per-instance scope on an evidence-trajectory signal, where neighboring work sits at composition- or trajectory-level on type-theoretic or risk-analytic signals.

0.10.7 9.3 Peircean inquiry and the philosophy of science

- Peirce 1878 (*Illustrations of the Logic of Science*, Popular Science Monthly): the original three-mode taxonomy of abduction, deduction, and induction.
- Peirce 1903 (*Pragmatism as the Logic of Abduction*, Harvard lectures): abduction as the only mode that introduces new content.
- Bacon 1620 (*Novum Organum*): induction as a typed primitive.
- Popper 1934 (*The Logic of Scientific Discovery*): falsification as the inductive-side constraint.
- Ramsey 1926 (*Truth and Probability*; in *The Foundations of Mathematics and other Logical Essays*, 1931): operational credence as betting odds, the Dutch Book argument, subjective probability as the substrate for graded belief. The hypothesis graph's node-level semantics descends from this work.
- James 1907 (*Pragmatism: A New Name for Some Old Ways of Thinking*) and Dewey 1929 (*The Quest for Certainty*): pragmatist commitment that truth is inseparable from action; the stakes-indexing of belief follows from this commitment.
- Meehl 1967: soft-science methodological critique that anticipates many of the failure modes our typing constrains.
- Feynman 1974 ("Cargo Cult Science"): informal but substantive on the difference between rigor-shaped activity and actual rigor.

0.10.8 9.4 Bi-abductive and compositional inference; failure isolation

- Calcagno et al. 2009: compositional shape analysis via bi-abduction; Facebook Infer as the industrial-scale instance of typed-mode inference on real code.
- Bylander et al. 1991: abductive computational complexity.
- O'Hearn 2019: separation logic and incorrectness logic, the modern compositional-inference scaffolding.
- Noam Zilberstein, Saliling & Silva 2024 (arXiv:2305.04842): Outcome Separation Logic; Theorem 5.1 establishes soundness of tri-abduction for branch composition under effects, extending bi-abduction from sequential to branching.
- Zeller & Hildebrandt 2002 (*Simplifying and Isolating Failure-Inducing Input*, IEEE TSE; building on Zeller 1999): delta debugging. An optimization-side adjacent: given a failure-inducing input, isolate the minimal failure-inducing subset by binary-search-shaped perturbation. The hypothesis graph in this work is

methodology-shape; delta debugging is optimization-shape; both rely on the reproducible / deterministic / perturbable properties of code (§3.1). Worth citing as the canonical demonstration that mechanical perturbation of code is a productive inference primitive.

0.10.9 9.5 Anytime-valid inference and evidence trajectories

- de Finetti 1937: subjective probability foundations.
- Ville 1939: martingales as the formal substrate for sequential evidence.
- Wald 1947: sequential testing of statistical hypotheses; the original "evidence is time-indexed" insight.
- Robbins 1967; Chernoff 1959: sequential design and stopping rules.
- Kelly 1996: capital growth and the betting-theory tie to e-values.
- Vovk & Wang 2021 (*E-values: Calibration, combination, and applications*, Annals of Statistics): the framework for e-values as non-negative random variables with $E_{H_0}[E] \leq 1$, calibration of p-values to e-values, combination, and anytime-valid stopping by Ville's inequality. The temporal-evidence stance in this paper (§3.4) borrows vocabulary from the sequential-evidence-trajectory framing; v1 deploys no formal e-value accumulator (the IID assumption SWE-bench audits violate would be required), and the paper makes no claim that e-values are epistemically preferable to p-values. That is not its venue.
- Shafer 2021: the *betting* interpretation of e-values; an e-value as the wealth of a strategy that bets against the null.
- Grünwald 2024; Ramdas 2023: modern syntheses of anytime-valid inference, e-processes, and safe testing.

0.10.10 9.6 Dynamical classification

- Milnor 1985; Strogatz 2014: dynamical-systems literature; source of the four shape words (convergent / divergent / oscillatory / chaotic), borrowed here as vocabulary only. The dynamical-systems treatment of evidence trajectories is developed in separate work.

0.10.11 9.7 Directed acyclic graphs as reasoning representation

- Pearl 1988 (*Probabilistic Reasoning in Intelligent Systems*): Bayesian networks as DAGs of probabilistic dependencies. The foundational application of DAGs to structured belief representation.
- Pearl 2000/2009 (*Causality: Models, Reasoning, and Inference*): structural causal models, d-separation, do-calculus. DAGs as the substrate for causal-structure inference.
- The hypothesis graph in this work adapts Pearl's representation primitive for a different purpose: typed *hypotheses about engineered systems under inquiry*, not probabilistic dependencies between random variables or causal relations between exposures and outcomes. We borrow the typed-node/typed-edge form and depart on two counts: the semantics over it are not Pearl's, and we relax acyclicity. Pearl's DAG is acyclic by definition, and acyclicity is what licenses d-separation; permitting cycles is a structural departure, motivated by cyclically-causal domains (SRE-style cascading failure) that an acyclic structure cannot represent faithfully.

0.11 10. Future Work

Four directions follow from this work. A held-out submission under the same artifact, one-shot discipline. Cross-instance smem accumulation: letting the hypothesis graph grow across instances within a repo, then across repos within a domain; the current work tests the smem at per-instance scope. A clean-room ablation on post-cutoff instances (SWE-rebench), with vs without the typed-mode constraint on one fixed model, to isolate the loop's effect on the rate. Skill-level retros: which stages of the loop carry which kinds of wins, and targeted skill freezes for follow-on benches.

The methodeutic-harness IR is not SWE-bench-specific. Any benchmark with falsifiable predicates and deterministic per-instance verdicts (HumanEval, MATH, theorem-proving suites, ARC, formal verification tasks, structured-output extraction) admits the same shape: abduction generates candidates, deduction derives an intervention, induction runs the predicate, the hypothesis graph carries belief and trajectory, the deterministic gate routes on shape. SWE-bench is the workload demonstrated here; the IR ports.

0.12 11. Availability and Reproducibility

- Repositories. github.com/kimjune01/swebench-pro and github.com/kimjune01/swebench-verified.
- Frozen artifact. Git tag `prereg-pro-v1`, SHA committed in the worklog.
- Preregistration. `PREREGISTRATION.md` at the freeze SHA.
- Provenance artifacts. Per-instance trajectories, hypothesis graphs, captured diffs, gate traces, and cost ledger under `runs/scored/artifacts/`.
- OSS deployment trace. Public corpus of ~380 hypothesis graphs at `kimjune01/sweep/repo-hypotheses/`, one per investigated issue across 46+ repositories. PR-level outcomes (merged / closed-unmerged / open) are pinned at `kimjune01/kimjune01@paper-2026-05-28` (the profile README itself drifts; the pinned commit is the citable artifact).
- OSS metrics are recomputable, not asserted. Every numeric claim in the deployment trace (merge rate, repo count, issue reception) ships alongside the GitHub GraphQL query that produces it. A skeptic with any GitHub token can paste the query into the GraphQL explorer and recompute the number in under a minute. No statistic is presented without its verifier. This is the same anti-grift mechanism as the bench's per-instance provenance, ported to the wild-deployment surface.
- Replication budget. ~$3/instance Sonnet-side × Pro bench size (~$2.2k, generator side; challenger side adds the vendor's published rate), or a single consumer subscription plus patience. See §4.5.
- Companion textbook. A reader-facing synthesis of the same dispersed lineage is available at june.kim/reading/methodeutics. The paper's argument rests on the primary sources cited in §2 and §9, not on the synthesis; the textbook is a navigation aid, not authority.
- Public-provenance trail. Dated blog posts at june.kim establishing the framework: *Theory is load-bearing* (2026-03-17), *The proof manual* (2026-04-05), *Type the question* (2026-04-08), *Evidence has a trajectory* (2026-04-27), *The hypothesis graph* (2026-04-28), *Abduction* (2026-05-04), *Modes of reason* (2026-05-04). These predate CMM (2026-05-26) and ADI (2026-04-17) and establish parallel rather than derivative development.
- PDF. Arxiv-shape build at /assets/methodeutic-harness-paper.pdf. Rebuilt from the markdown source by `scripts/build-paper-pdf.sh`; the source is canonical.
- DOI. [placeholder: Zenodo paper-DOI distinct from the software-DOI.]
- License. Skills are released free and openly under CC-BY-SA-NS (see june.kim/cc-by-sa-ns). Repo-level terms in `LICENSE.md`. The harness, the skills, the trajectories, the hypothesis graphs, the gate logic: all freely available for inspection, replication, modification, and reuse under attribution + share-alike + no-spam terms. No paywall, no gated access, no enterprise tier; the harness an outsider clones is the same harness that produced the published numbers.

Reproducibility invitation. *Nullius in verba.* The repository, per-instance trajectories, hypothesis graphs, gate traces, captured diffs, and cost ledger are public. The harness runs end-to-end on a frontier-vendor subscription or under ~$2k of API spend per bench (§4.5). Replication does not require institutional credentials or enterprise access. Doubts about specific instances, regrades, or methodological claims should be filed as issues against the repository; confirmed corrections fold into the next versioned artifact.

0.13 LLM Collaboration Disclosure

Per current ethics norms for AI-assisted scientific writing, the use of LLMs in this work is disclosed here in two distinct roles.

Subject of study. The harness under evaluation (§3, §4) uses frontier LLMs as the generator and challenger inside the methodeutic harness; this is the paper's object of inquiry rather than a writing aid, and the specific model versions, billing mode, and provenance are fully disclosed in §4.3 and the published artifact.

Writing aid. This paper was drafted with assistance from Anthropic's Claude (Opus 4.7) for converting a human-authored bullet-point outline into prose, for editorial passes (sharpening, tightening, em-dash and negative-parallelism linting), and for surface-form copyediting. All technical claims, citations, numeric results, methodology design, and the structure of the argument originate with the human author; LLM assistance was confined to surface-form editing of human-authored content. No LLM acted as a peer reviewer, determined any of the paper's claims, or selected what to publish. Every paragraph was read and approved by the human author before commit.

0.14 Acknowledgments

-