Dual-Axis Benchmark and Preventive Scaffold: Surfacing and Mitigating Silent Risks in LLM Data Preparation

Ming Chen and Shaikh Arifuzzaman
Department of Computer Science
University of Nevada, Las Vegas
Las Vegas, NV, USA
Emails: chenm24@unlv.nevada.edu, shaikh.arifuzzaman@unlv.edu

***Abstract*—Large language model (LLM) agents working on raw, heterogeneous data may fail silently when they act on upstream assumptions that turn out to be wrong. Existing evaluations rarely surface this risk, benchmarking on clean, pre-identified inputs; agentic frameworks respond by layering self-critique or multi-agent debate above the executor. We introduce DiscRun-Bench, a benchmark whose family/variant design decomposes hazard *discovery* from hazard-aware *execution* and scores data correctness separately from caveat-surfacing diligence. We then treat PRVR (Plan-Run-Verify-Resolve), a scaffold imitating industry-standard plan-execute-verify scaffold, as a measurement probe. Structuring Claude Haiku 4.5's operating loop to ask about hazards lifts its caveat-surfacing rate from around 0.74 to 0.90 and its mean task score by roughly 25%, both significant under paired permutation tests. Attribution assigns the gain to the preventive Plan stage and to delivery-layer robustness, not to the reactive self-critique retry. PRVR-Haiku meets or exceeds a bare Sonnet 4.6 reference at roughly $4\times$ lower per-task list-price cost, decomposing into a $3\times$ per-token price ratio and a roughly $1.35\times$ grid-mean token ratio driven by a few high-token cells. Silent failure in LLM data preparation is thus substantially an *elicitation* problem rather than only a capability problem.**

***Index Terms*—large language models, data preparation, agent benchmarks, verification, uncertainty, evaluation**

I. Introduction

Large language models (LLMs) have demonstrated strong ability across a wide range of tasks, from natural language understanding to multi-step reasoning and code generation. One emerging application is data preparation, the stage that transforms raw, heterogeneous source files into analysis-ready inputs for downstream modeling [1]. LLM agents equipped with tool use have made real progress here: by keeping large intermediate values in the file system rather than in the conversation, modern systems partially sidestep the context-window cliff, since megabytes of tabular data never enter the prompt.

Even so, several failure modes persist: brittle reasoning over long replayed contexts [2], plausible but unverified generated code [3], and confidence decoupled from correctness [4].

Among these, the most consequential is *silent assumption-failure*: the agent produces output that looks correct but rests on an upstream assumption that is wrong. A directly-wrong answer prompts immediate inspection. The test fails, the pipeline halts, someone investigates. A silently wrong answer, by contrast, flows downstream unchallenged, often surfacing only after weeks of model training, analysis, or reporting have already been built on top of it. The failure that does not crash surfaces downstream as model performance drift, mismatched joins, or analyst error blamed on the wrong cause.

Consider an LLM asked to compute monthly returns for the S&P 500 from a membership file that stores each date with a comma-separated string of tickers rather than the more common one-column-per-ticker layout. Assuming the common form, the model exhausts its attempts rewriting parsers and the run fails visibly, with no output. Had it instead quietly extracted only the first ticker per row, it would have produced a normal-looking table tracking one stock rather than the full index: the same root error, hidden inside a passing output.

With the audit trail back to the preparation stage rarely intact, such silent failures make the field worth benchmarking specifically for an agent's ability to surface its own uncertainty, not only for its end-to-end correctness rate. A failure is silent precisely when no caveat accompanies the output; the rate at which an agent surfaces the applicable caveats is therefore the measurable complement of silent failure, and it is the quantity our evaluation tracks alongside correctness. Industry is increasingly acknowledging this issue as well. Anthropic's recent Claude Opus 4.8 release, for example, highlights the model's improved ability to flag items it is uncertain about for human review, rather than silently and confidently passing them along the pipeline as correct [5].

Existing benchmarks for LLM data work measure outcomes after a forward pass over clean, pre-identified inputs [6]–[8]: they answer whether the model can compute X given Y, but not whether the model would have noticed Y's hidden risks if no one named them. Two competencies that are essential to safe data preparation, *discovering* latent hazards and *executing* on hazards the user has named, are conflated in current evaluation.

Our three contributions are organized around a single question: on practical data preparation tasks, do agents fail to surface latent hazards because they cannot, or because nothing

in their operating loop asks them to?

(i) DiscRun-Bench (Sec. III): a benchmark whose family/variant design decomposes discovery from execution and scores data correctness independently from caveat-raising diligence; ten families and thirty-three variants exercise the failure-mode codes of Table I across a documentation-utility axis.

(ii) Cross-model findings (Sec. III-E): explicit prompt hints help on two orthogonal axes, directing attention to the named hazard and narrowing the construction path, the second invisible to standard scoring. Across four models, the same failure label further hides distinct regimes, including a delivery-layer regime in which an otherwise acceptable answer is lost to a formatting accident at submission.

(iii) An elicitation finding (Sec. IV): we use PRVR (Plan-Run-Verify-Resolve), a scaffold probe that instantiates the plan-execute-verify pattern common in deployed data-agent stacks [1], [9], [10], to test whether the missing caveats can be elicited. Structured to ask, Haiku 4.5 surfaces most of the hazards it previously missed: its caveat-surfacing rate rises from 0.74 to 0.90 alongside a roughly 25% score lift, and transcript-level attribution places the gain in the preventive Plan stage and in delivery-layer robustness rather than in the reactive self-critique retry.

II. Related Work

This work relates to three lines of research: LLM-agent benchmarks for code and data tasks, verification and self-critique mechanisms for LLM agents, and classical data quality and preparation.

A. LLM-Agent Benchmarks for Code and Data

Existing benchmarks for LLM-based data work largely assume that the relevant files are pre-identified and supplied in clean, well-specified form. DS-1000 [6] measures code generation against curated problems with explicit specifications and reference inputs. DA-Code [7] and DataSciBench [8] extend to agentic settings but retain the clean-input assumption: the agent is told which files to read and what they contain. DABstep [11] extends similarly to multi-step data agents. KramaBench [12] is closest to our scope; it explicitly targets data lakes and observes that existing benchmarks do not capture "the heterogeneity of data tasks" or the demands of "large, domain-specific, and unclean input datasets."

Existing benchmarks cover important parts of the LLM data-workflow problem, but they usually evaluate the agent after the task framing has already identified the relevant inputs, goal, or cleaning operation. As a result, they primarily measure whether an agent can execute a specified data task, rather than whether it can independently notice that the task setup contains a latent data risk. This distinction matters for realistic data preparation because many high-impact failures are not syntax errors or direct computation mistakes; they are unreported assumptions about units, coverage, temporal validity, schema shape, or source reliability.

Our DiscRun-Bench (Sec. III) addresses this gap by scoring hazard discovery separately from execution on matched underlying data, with prompt specificity and documentation visibility varied as independent axes.

B. Verification and Self-Critique in LLM Agents

LLM agents reduce single-shot error rates through prompting techniques such as chain-of-thought reasoning [13], self-consistency sampling [14], and iterative self-critique frameworks including Reflexion [15] and Self-Refine [16]. These methods improve accuracy on closed-form problems but treat verification as an unstructured prose step: the agent is asked to "check its work" without a forced output schema or per-decision provenance. In multi-step data processing this is brittle: the agent can produce plausible confirmations of incorrect intermediate state, and errors compound silently across stages with no gate to surface them.

A parallel line of work develops LLM-based data-science agents: AutoPrep [1] translates natural-language data questions into preparation pipelines via a multi-agent framework; DS-Agent [9] and Data Interpreter [10] plan and execute multi-step data workflows over heterogeneous inputs. These systems demonstrate the value of multi-stage decomposition but do not structurally enforce uncertainty surfacing: agent outputs remain free-form prose, and the confidence behind each decision is not separated from the decision content. Empirical work on confidence elicitation [4] further shows that LLMs are systematically overconfident, with the effect most pronounced in tasks requiring specialized domain knowledge, precisely the regime in which data preparation operates. This plan-execute-verify skeleton is now the de facto operating pattern of deployed data-agent stacks. PRVR (Sec. IV-A) instantiates it unchanged, save a deterministic format gate on Verify, not as a competitor to Reflexion or Self-Refine but as an instrument for measuring which component of the pattern elicits uncertainty; the reactive self-critique retry contributes least, roughly 6% of the lift (Sec. IV-C).

C. Data Quality and Preparation

Classical work on data quality has long catalogued the hazards we target in this paper: schema drift, unit inconsistency, leakage across temporal joins, ambiguous missingness semantics, and survivorship bias in panel data [17], [18]. Interactive tools such as OpenRefine [19] and Trifacta-style wranglers [20] operationalize this knowledge as human-in-the-loop workflows. More recent data-validation libraries [21] encode quality rules as executable assertions over pipelines. Our work inherits problem definitions from this literature but treats them as *measurement targets* for LLM agents rather than as tasks for manual cleaning or human-authored assertions.

III. DiscRun-Bench: Design and Findings

DiscRun-Bench is designed around the distinction the Sec. I question implies: an agent's ability to surface latent data hazards (discovery) is separable from its ability to act on

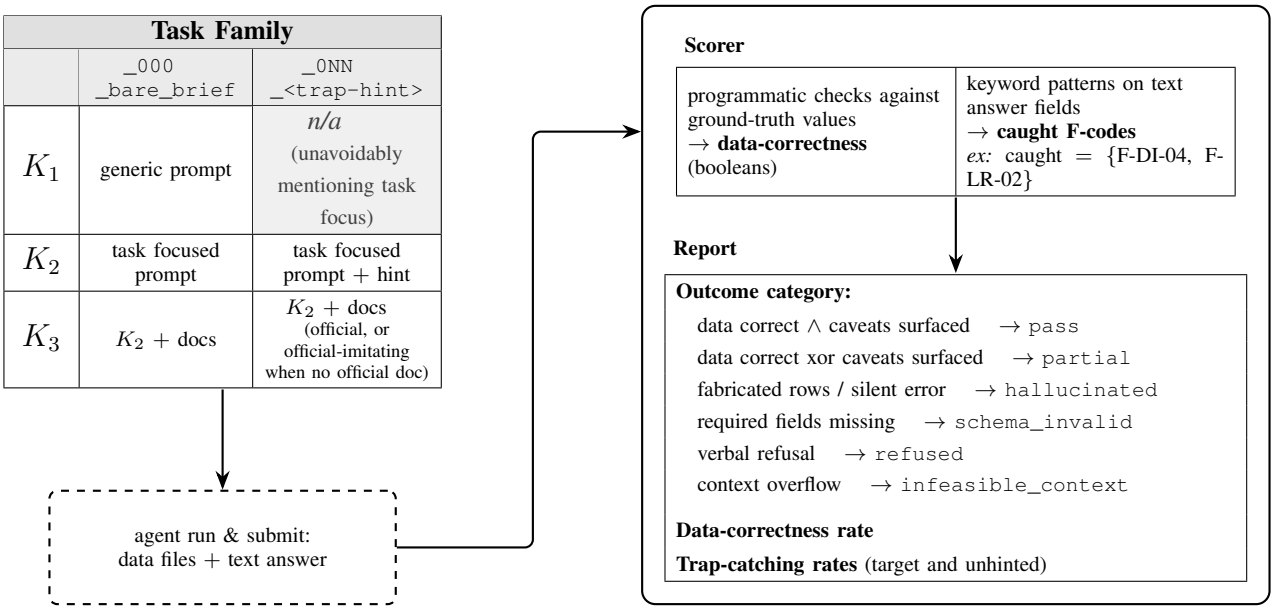

Fig. 1. Overview of DiscRun-Bench. Scoring-set definitions (catchable, hinted, target, caught) and rate formulas: Sec. III-C.

a hazard the prompt has named (execution), and the benchmark scores the two independently. It measures realistic data-preparation work: from raw source files and a downstream goal to an analysis-ready output. Evaluation combines programmatic checks against ground-truth values with keyword pattern matching on prose answer fields for trap-catching detection (similar to how DS-1000 [6] combines execution tests with keyword pattern matching on the agent's generated code). Both layers are deterministic; no LLM-as-judge is used. Compared with prior benchmarks (Sec. II), DiscRun-Bench differs along three axes that separate discovery from execution and measure each at multiple documentation levels. The suite spans ten datasets across six domains (Table II). Tasks examine skills in seven groups drawn from data-preparation taxonomies [18]: ingestion, schema discovery, cleaning and quality, normalization, joining and aggregation, analysis and inference, and validation and documentation. Figure 1 overviews the experimental design and the scoring pipeline.

A. Family and Variant Design

Each task in DiscRun-Bench is a *family* of variants sharing the underlying dataset and catchable hazards but differing in prompt and pass criterion. A family folder `T<NNN>_<slug>/` contains a reserved `_000_bare_brief` slot (the no-hint baseline) and one or more `_0NN_<trap-hint>` sister variants. Two family styles are in use. In *Style A (deployment-archetype)* families the trap inventory is constant; sister variants differ in the intensity of the hint and the required shape of the response (e.g., halt-on-error vs. skip-and-report vs. autonomous-discovery). In *Style B (per-trap-hint)* families the trap inventory has multiple distinct hazards and each sister variant hints exactly one, decomposing per-trap discovery (in bare-brief) against per-trap execution (in the matching sister variant).

Every variant's expected output declares three sets of failure-mode codes (defined in Sec. III-B): *catchable* (the real traps in the data, constant across the family), *hinted* (codes the prompt directs the agent toward; empty for bare-brief, one code for hint variants), and *target* (codes the pass criterion requires the agent to surface). The scorer emits a fourth set at runtime: *caught*, computed by mapping per-task scorer signals (booleans recording whether the agent surfaced each hazard in its produced data or its prose answer) to failure-mode codes. The four sets feed three caught-rate metrics (Sec. III-C).

Orthogonal to the variant axis is the *documentation-utility axis*: each variant can be run at one of three K-levels. K_1 supplies a universal generic prompt ("clean and prepare this data for analysis") over a minimal staged tree containing only the raw data files; the agent must discover structure empirically. K_2 supplies a variant-specific task prompt over the same minimal staged tree. K_3 uses the same variant-specific prompt as K_2 (byte-identical) over the full dataset folder, which adds a dataset notes file, format specs, and code dictionaries. Adjacent levels vary one factor at a time: K_1 to K_2 changes only prompt specificity, K_2 to K_3 only documentation visibility. For bare-brief variants the full K_1–K_3 axis is defined; for trap-hint variants K_1 is omitted: a generic prompt cannot carry the variant's response shape or trap focus. The K-levels each variant is actually collected at on the released grids are stated in Sec. III-D.

B. Failure-Mode Taxonomy

A cross-task taxonomy of failure-mode codes lets per-task scorers emit comparable diagnoses across families. Codes use the format `F-<CLASS>-<NN>` and group into five classes: data integrity (F-DI), format/parsing (F-FP), schema/output (F-SO), leakage and reasoning (F-LR), and refusal/capability (F-RC). Table I lists the currently active codes.

TABLE I
ACTIVE FAILURE-MODE CODES BY CLASS.

Code	Description
F-DI: data integrity	
F-DI-01	silent source drop
F-DI-02	fabricated rows
F-DI-03	prose-only caveat when structured required
F-DI-04	undeclared partial coverage
F-DI-05	cross-source disagreement undeclared
F-FP: format/parsing	
F-FP-01	unit unconverted
F-FP-02	bundled-doc format mismatch
F-FP-04	aggregate-metadata misreshape
F-SO: schema/output	
F-SO-01	required field missing or malformed
F-LR: leakage/reasoning	
F-LR-01	cell-level look-ahead
F-LR-02	universe-level survivorship bias
F-RC: refusal/capability	
F-RC-01	verbal refusal
F-RC-02	model lacks tool-use capability

Codes name kinds of failure rather than specific incidents. Rate denominators use only the catchable subset (real data traps the agent can surface), excluding agent-internal codes such as F-DI-02 and F-RC-01: the agent does not "catch" its own failures.

C. Dual-Rate Scoring

The scorer reports *data-correctness rates* (right values, checked against canonical references) and *trap-catching rates* (catchable hazards surfaced) for each run. For trap-catching, the *target-catching rate* (r_{tgt}) is the fraction of target codes the agent surfaced; the *unhinted-catching rate* (r_{unh}) is the fraction of catchable codes not named in the prompt that the agent surfaced. The headline rate depends on the variant slot: r_{unh} for bare-brief variants ("discovery without prompting"), and r_{tgt} for trap-hint variants ("execution when told"). The hint variant's r_{tgt} minus the bare-brief's r_{unh} measures how much the explicit hint helped.

Each run is also assigned exactly one *outcome category* from the Report block of Fig. 1. Reporting both rates and categories prevents two agents with comparable headline scores from being misread as equivalent: a "silent execution" agent (high correctness, zero trap-catching) and a "diligent but incompetent" agent (high trap-catching, wrong values) end up in different outcome categories despite both being failure modes. Many existing data-science benchmarks fold these distinctions into a single accuracy score [6], [11], [12], leaving failure causes to separate qualitative analyses rather than a per-run categorical breakdown reproducible across new runs.

D. Task Suite

Table II lists the ten task families that make up the suite previewed in the section opener. *Tier* is a task-complexity axis (L1 / L2 / L3, defined in Table II's caption) distinct from the per-family variant *Style* discussed in Sec. III-A. Every task ships with the ground-truth files its scorer compares against; the experimental subset used in this paper is identified in Sec. IV-B.

The task suite was assembled to span data-error classes (data integrity, parsing, leakage, schema) and domain conventions including scientific measurement, regulatory disclosure, transactional commerce, and ML preprocessing. The released grids cover all ten families; no family is held out. Every variant runs at K_2; bare-brief variants additionally run at K_1 on a five-family subset and at K_3 on a seven-family subset (the collected cells of Table III(a)); all trap-hint variants also run at K_3; and the Style-A autonomous-discovery variant, which names no trap, runs at K_1 as well. Section IV-B defines the two experimental grids drawn from this coverage.

E. Findings on LLM Behavior

The bench-baseline grid (Sec. IV-B) is run on four models under the `agent` scaffold, ordered by capability: Anthropic Claude Sonnet 4.6, Anthropic Claude Haiku 4.5, OpenAI GPT-4o-mini, and Google Gemini 2.5 Flash-Lite. The lineup spans three providers and a capability gradient on which the same outcome label can be probed for different underlying failure regimes (Sec. III-E3).

1) Hints could help on two axes: The expected effect of an explicit trap hint is to lift the target-catching rate r_{tgt}. A second effect, less visible to standard scoring, is on convergence: hints narrow the agent's construction path, so the executor reaches `submit_answer` with fewer wasted iterations and on more cells. On Sonnet 4.6 $\times$ T004 K_2, the bare-brief variant returns `partial` (score 0.56) after 19 iterations exploring the entity-resolution policy, while two trap-hint sisters pass with full data-correctness in 11 and 16 iterations. In aggregate, Sonnet passes 70 % of the 23 trap-hint variants at K_2 (Table III(b)), well above its bare-brief pass rate on the matched tasks. The convergence axis thus appears as both an iteration delta and a category lift, and decides whether the agent reaches a passing submission at all.

2) More documentation is not always more useful: K_3 isolates documentation visibility from prompt content (Sec. III-A). Reading across the Sonnet rows of Table III(a) shows K_3 helps on some tasks but hurts on others. On T006, the bundled docs coincide with recovery: Sonnet's failed K_2 cell passes at K_3. The mechanism is, however, confounded: the K_2 failure is partly a delivery accident (Sec. III-E3) in which a submission that surfaced the target caveat was wrapped in an extra envelope layer and scored schema-invalid, so the recovery cannot be attributed to the documentation alone. On T008, by contrast, the added documentation pushes the agent past its iteration cap on a multi-source merge, flipping a K_2 pass to a K_3 schema-invalid outcome. This pattern aligns with prior observations that longer context does not automatically improve task performance [2], [22]: extra documentation raises the agent's exploration cost in the loop, risks exhausting the context limit or iteration budget, and can degrade focus

TABLE II

TASK SUITE SUMMARY. TIER: TASK COMPLEXITY (L1 = ATOMIC, L2 = COMPOUND, L3 = CROSS-DATASET; L2/L3 = BOTH). VARIANTS: VARIANTS PER FAMILY, INCLUDING THE BARE-BRIEF SLOT. HEADLINE CATCHABLE TRAPS: SCORING DENOMINATORS FOR THE BARE-BRIEF VARIANT.

ID	Domain	Dataset family	Tier	Variants	Headline catchable traps
T001	Hydrology	USGS NWIS streamflow	L2	4	F-DI-01, F-FP-02
T002	Climate	GHCN-D + GISTEMP	L2	4	F-FP-01, F-FP-04, F-DI-04
T003	Finance	FRED macro series	L2	3	F-LR-01, F-DI-04
T004	Finance	PPP loan registry	L2	4	F-DI-01, F-DI-04
T005	Finance	S&P 500 historical	L2	3	F-LR-02, F-DI-01, F-DI-04
T006	Public health	Chicago food inspection	L2	3	F-DI-04
T007	Social science	World Bank WDI	L1	3	F-DI-01
T008	Social science	WDI + Gapminder	L2/L3	3	F-DI-05, F-FP-04, F-DI-04
T009	ML datasets	8 UCI/sklearn tabular sets	L2/L3	3	F-DI-04
T010	Retail	UCI Online Retail	L2	3	F-DI-04

TABLE III

BENCH-BASELINE OUTCOMES ON THE AGENT SCAFFOLD, SEED 0 PER CELL (THE COMPARISON GRID OF SEC. IV-B RUNS $N=3$ SEEDS). **(A)** BARE-BRIEF OUTCOME BY K-LEVEL; CODE KEY: P (PASS, SHADED), p (PARTIAL), s (SCHEMA_INVALID), I (INFEASIBLE_CONTEXT), H (HALLUCINATED), **–** (NOT COLLECTED: RELEASED K_1/K_3 SUBSETS, SEC. III-D). BOTTOM ROWS: K_1 BARE-BRIEF PASS COUNT (OF 5) AND MEAN K_2 EXECUTOR ITERATIONS. **(B)** TRAP-HINT SISTERS OF SEC. III-A BY OUTCOME CATEGORY (%, $n=23$ PER CELL); K_1 OMITTED BY DESIGN.

(a) Bare-brief outcomes (no per-trap hint)

	Sonnet 4.6			Haiku 4.5			4o-mini			Flash-Lite		
Task	K_1	K_2	K_3	K_1	K_2	K_3	K_1	K_2	K_3	K_1	K_2	K_3
T001	–	P	–	–	P	–	–	s	–	–	s	–
T002	h	P	P	h	P	P	i	s	s	s	s	s
T003	–	P	–	–	P	–	–	s	–	–	s	–
T004	P	p	P	P	h	s	s	p	s	s	s	s
T005	–	P	P	–	P	P	–	s	i	–	s	s
T006	p	s	P	p	p	s	p	p	p	s	s	s
T007	–	P	–	–	p	–	–	s	–	–	s	–
T008	s	P	s	s	P	p	s	s	s	s	s	s
T009	–	P	P	–	p	P	–	i	s	–	s	s
T010	p	p	P	P	p	p	s	s	s	s	s	s
K_1 bare-brief pass	1/5			2/5			0/5			0/5		
mean n_{iter} at K_2		15.3			11.4			19.1			2.7	

(b) Hint variant result distribution (%)

	Sonnet		Haiku		4o-mini		Flash	
	K_2	K_3	K_2	K_3	K_2	K_3	K_2	K_3
pass	70	57	57	57	9	9	0	0
partial	22	30	22	22	9	13	4	0
schema	9	9	17	22	65	65	96	100
infeas.	0	0	0	0	17	9	0	0
halluc.	0	4	4	0	0	4	0	0

while exploring it. A separate failure mode appears on 4o-mini × T006 K_3: the agent echoes documentation vocabulary into its explanation without matching the policy in code. Documentation utility therefore depends on the role the docs play for each task, not on the K-level alone.

3) Same label, different mechanisms: Reading the columns of Table III(a) left to right gives a capability gradient, but each model's failures look different. Sonnet passes most tasks (70% hint-variant pass rate). Haiku 4.5 mixes passes, partials, and the occasional hallucination at a mean of 11.4 iterations per task and a 57% hint-variant pass rate, the operating point the scaffold probe of Sec. IV targets. 4o-mini grinds without converging: a mean of 19.1 iterations and a $\sim$65% schema-invalid rate on hint variants, with failing cells running near the 30-iteration cap and a smaller infeasible-context band traced to context overflow. Flash-Lite, by contrast, gives up early: a mean of 2.7 iterations and a 96–100% schema-invalid rate; it emits one or two exploratory tool calls, stops planning, and the K-axis has no effect on outcome.

A further regime hides inside the shared label. In 22 of 147 bare-Haiku runs (15%), the agent submits an otherwise acceptable answer wrapped in one extra envelope layer; the controller's strict one-layer unwrap then scores the run schema-invalid. The other models essentially never hit this delivery-layer failure (2 of 148 runs for Sonnet, 0 of 69 for 4o-mini, 0 of 99 for Flash-Lite), so Haiku's schema-invalid column mixes delivery accidents with genuine capability failures. Misses on the two capable models are likewise hazard-structured: unit-conversion hazards are almost always surfaced (0 of 19 and 1 of 19 missed for Sonnet and Haiku), bundled-documentation format mismatches almost never (18 of 19 and 18 of 18). The same outcome label thus covers five failure regimes: task-difficulty ceiling, mid-tier capability mix, context-window limit, tool-use stamina, and delivery-layer fragility. Lifting the bare baseline must restore the convergence axis without the task-specific knowledge hints carry; Sec. IV-A sets up that probe.

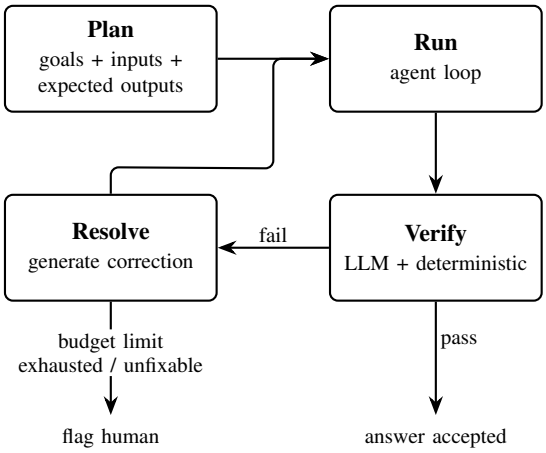

Fig. 2. The PRVR cycle architecture.

IV. The PRVR Harness

A. Design

PRVR (Plan-Run-Verify-Resolve) is a four-phase harness that wraps the same tool-using agent loop, tool surface, model, and per-Run iteration ceiling as the bench-baseline controller of Sec. IV-B; only the cycle of Fig. 2 differs, so the comparison in Sec. IV-C measures the cycle's effect rather than differences in the executor or the bookkeeping. Instantiating the standard plan-execute-verify pattern of Sec. II with no architectural novelty claimed, PRVR serves as a measurement probe: if structuring the operating loop to ask about hazards is enough to surface them, the missed caveats of Sec. III-E reflect an elicitation gap rather than a capability ceiling. Within the cycle, Plan reads the task brief once and emits an *advisory* orientation (goals, candidate input files, and answer topics) that Run is told to override when the data contradicts it; Run is the unmodified bench-baseline agent loop with the orientation appended to its system prompt and corrections appended on retry, terminating on `submit_answer` or the iteration cap. Verify reads the submitted answer alongside a deterministic per-file format check whose `[FAIL]` forces the verdict to fail regardless of the LLM's judgment; Resolve either retries with corrections that re-enter Run or terminates the cycle by flagging a human.

The scaffold is task-agnostic: the same code runs on every variant in the comparison grid, and Plan and Verify receive no per-task configuration beyond the task prompt itself and the required answer-field names that the prompt already declares to the bench-baseline agent.

B. Experimental Setup

Two experimental scopes share the same task suite (Sec. III) and reporting conventions: a bench-baseline grid for cross-model failure analysis (Sec. III-E) and a PRVR comparison grid for the head-to-head on Claude Haiku 4.5 (Sec. IV-C).

1) Grids and models: The bench-baseline grid runs all ten task families of Table II, across their full variant sets and the relevant K-levels (Sec. III-A), on the bare agent scaffold for four models ordered by capability: Claude Sonnet 4.6, Claude Haiku 4.5, GPT-4o-mini, and Gemini 2.5 Flash-Lite; results feed Sec. III-E. The PRVR comparison grid pairs bare and PRVR runs on Haiku 4.5 over every K_2 variant of Table II (bare-brief and all trap-hint), the five bare-brief K_1 slots, and the Style-A autonomous-discovery variant. Selected K_3 cells are run for exploratory work and verify PRVR matches the higher bare-baseline pass rates at K_3.

2) Seeds, significance, and the delivery-normalized baseline: Every condition on the comparison grid runs $N = 3$ seeds per cell: bare Haiku and PRVR-Haiku, which are fully paired, and the bare-Sonnet reference alike. Reported cell values are seed means; the bench-baseline grid of Sec. III-E displays seed 0. Significance for cell-level comparisons is assessed on the paired cell deltas with a sign-flip permutation test and a Wilcoxon signed-rank test, reported alongside the corresponding means in Sec. IV-C. Finally, to separate what the scaffold elicits from what its submission handling repairs, we define a *delivery-normalized baseline*: bare submissions that failed only by the extra envelope layer of Sec. III-E3 are re-scored offline with the tolerant answer extraction the PRVR controller applies, with no new model calls. Sec. IV-C reports this counterfactual as a third condition between bare and PRVR.

3) Metrics and cost reporting: Each run is summarised along three metric classes: outcome category (Fig. 1); trap-catching rates (r_{tgt}, r_{unh}, and the analogous all-catchable rate r_{all}); and convergence telemetry (iteration count and token usage). Reported aggregates use the headline-rate-per-variant-type convention of Sec. III-C. Cost is reported as total tokens per task in millions (Mtok), summing input, output, cache-read, and cache-creation token counts across all LLM calls made for a task; this is cache-invariant and provider-agnostic. Raw billed cost would be cache-asymmetric on the released grid (the bare controller benefits from prompt caching; the current PRVR controller does not yet mark its prompts for it). Where a billed-cost interpretation aids deployment context (Sec. IV-C3), we report list-price cost: per-task token consumption multiplied by each model's published per-token prices.

C. PRVR Findings

Table IV summarizes the comparison grid as three conditions (bare, delivery-normalized bare, and PRVR) alongside the bare-Sonnet reference. This subsection asks three questions in turn: whether the probe lifts caveat surfacing, where its score lift comes from, and what it costs relative to upsizing the model. Cross-model transfer is deferred to Sec. V-B.

1) The probe lifts caveat surfacing: Across the comparison grid, the probe lifts Haiku's headline caveat-surfacing rate (Sec. III-C) from 0.738 to 0.898, a gain of 0.159 with 16 wins, 19 ties, and 3 losses at $|\Delta| > 0.05$. The lift is significant under a paired cell-level sign-flip permutation test ($p = 2.4 \times 10^{-4}$) and a Wilcoxon signed-rank test ($p = 7.3 \times 10^{-4}$), and it appears on both benchmark axes: the unhinted (discovery) rate rises from 0.571 to 0.739 and the

TABLE IV
COMPARISON-GRID SUMMARY (39 CELLS, $N=3$ SEEDS PER CONDITION). CAUGHT = HEADLINE CAVEAT-SURFACING RATE ($n=38$; ONE T004 VARIANT DECLARES NO TARGET CODE BY DESIGN). DELIVERY-NORMALIZED = THE OFFLINE COUNTERFACTUAL OF SEC. IV-B2: ITS SCORE SPANS THE BOUNDS; – MARKS RATES NOT MEASURABLE FROM ARCHIVED OUTPUTS.

Condition	Score	Caught	Pass	Mtok
bare Haiku 4.5	0.71	0.74	0.46	0.32
+ delivery normalization	0.74–0.84	–	–	0.32
PRVR Haiku 4.5	0.90	0.90	0.62	0.37
bare Sonnet 4.6	0.82	0.85	0.67	0.50

target (execution) rate from 0.759 to 0.934. It is also robust to the prompt-vocabulary exclusions of Sec. IV-A, remaining between +0.125 and +0.165 across the exclusion sets. The scaffolded small model surfaces more than the bare-Sonnet reference does (0.898 against 0.852). Much of the attention-axis benefit of explicit hints (Sec. III-E1) is thus elicitable without task-specific knowledge.

2) Where the score lift comes from: On the same grid, PRVR lifts mean task score from 0.71 to 0.90, an increase of 0.181 absolute and 25.4% relative (19 wins, 16 ties, 4 losses at $|\Delta| > 0.05$; permutation $p = 1.6 \times 10^{-4}$, Wilcoxon $p = 7.6 \times 10^{-4}$). The lift is sharper at K_1 (six cells: 0.69 to 0.95), where the prompt names neither trap nor documentation, than at K_2 (33 cells: 0.72 to 0.89), where the bare baseline already converges more often. The wins concentrate on cells where the bare agent exhibits high per-seed variance or schema-invalid failures (almost every win cell has bare-seed standard deviation above 0.2 or bare mean below 0.7); cells where bare-Haiku already converges reliably appear as ties that pay the cycle's overhead without changing the outcome.

The +0.181 lift decomposes into four channels. Plan-helped one-shot convergence contributes +0.103 (57%): the pre-committed orientation makes the first guess informed rather than improvised. Tolerant answer extraction contributes +0.087 (48%): PRVR's submission handling alleviates the delivery-layer failures of Sec. III-E3; offline re-scoring is possible for around 30% of baseline delivery failure seeds (the rest was lost due to size), and 50% of those recover. The Verify-Resolve retry contributes +0.011 (6%) and regression cells drag -0.019 (-10%).

PRVR's regression cells divide into two modes: Plan correctly identified a decision point but delegated it to the executor without a clear commitment, or Plan named the constraint, the agent failed to follow it, and the Verify-Resolve cycle did not catch the deviation. Both are consistent with the weak retry channel: the cycle's reactive half rarely recovers what its preventive half misses.

3) Cost against upsizing the model: Against its own bare baseline, PRVR's per-cell token premium averages $1.36\times$ (range $0.60\times$ to $3.99\times$; the upper end on halt-on-error variants, where the bare baseline runs only a handful of tool calls and the Plan-plus-Verify overhead dominates). Against upsizing, PRVR-Haiku scores 0.90 to the bare-Sonnet reference's 0.82 while consuming 0.37 Mtok per task to Sonnet's 0.50; strict per-cell wins at $|\Delta| > 0.05$ split 13 PRVR-Haiku, 21 ties, and 5 Sonnet. The roughly $4\times$ list-price gap (methodology in Sec. IV-B3) decomposes into a price factor and a token factor: Sonnet is priced $3\times$ higher than Haiku per token on both input and output, and the grid-mean token ratio is roughly $1.35\times$, itself driven by a handful of high-token Sonnet cells (notably T005) while per-cell consumption is near parity (cell-mean token ratio 0.98). Under the same paired cell-level tests, the score advantage over the reference is weaker than the head-to-head (permutation $p = 3.5 \times 10^{-2}$, Wilcoxon $p = 5.8 \times 10^{-2}$), consistent with the tie-heavy split. Sonnet's own two envelope failures are seed-runs of the T006 bare-brief cell noted in Sec. III-E3; peeled offline they score a low partial (0.062) with the target caveat surfaced, leaving the grid comparison essentially unchanged.

V. DISCUSSION AND LIMITATIONS

A. Implications

Two implications follow for evaluation. First, scoring caveat surfacing separately from correctness separates *elicitation* gaps from *capability* gaps: most of the diligence the bare small model left unexpressed was elicitable by loop structure alone (Sec. IV-C1), a distinction a single accuracy number cannot draw. Second, delivery-layer failures are a distinct regime, not a small-model tax: a substantial share of apparent capability shortfall is delivery fragility (14% to 71% of the head-to-head lift), and a scaffold repairs it almost for free.

For deployment, the results support a scaffold-before-upsize decision rule: when a deployment's failures concentrate in recoverable formatting and unsurfaced caveats, structure the operating loop before paying for a larger model. On the released grid, PRVR-Haiku exceeds the bare-Sonnet reference on mean task score at roughly $4\times$ lower list-price cost per task (Sec. IV-C3). The rule's scope is visible in the same data: the advantage concentrates in low-hint tasks, while on more-hint tasks PRVR-Haiku and bare Sonnet are nearly tied (0.65 vs 0.72), as expected when the hazard is already adequately hinted. The cheapest robustness lever the study surfaces is one line of tolerant answer parsing at the submission boundary; it repairs the delivery accidents of Sec. III-E3 at zero model cost, though not the cells where the wrapped submission was also wrong.

B. Cross-model transfer limit

Two cross-model probes (Gemini 2.5 Flash-Lite and GPT-4o-mini) test whether PRVR's lift transfers to weaker models (Sec. IV-B). Neither gains, for opposite reasons. Flash-Lite gives up at two iterations under the bare baseline and continues to wander under PRVR's advisory Plan: weak instruction-following cannot responsibly use orientation that the agent is told to override when the data contradicts. GPT-4o-mini runs the full iteration cap and exhausts its 128K context window on accumulated tool-call output before reaching a clean submission. Any harness layered on top inherits both limits: advisory

planning presumes instruction-following judgment, and multi-source tasks presume a context budget; broader cross-model validation and per-model scaffold tuning are future work.

C. Limitations

Several limitations bound the claims. *Probe scope.* The elicitation finding is established on one model family: the paired head-to-head runs on Haiku 4.5 only, and the cross-model probes above show the probe is not free-standing on weaker models. Whether the same gap exists in other mid-tier models, and whether PRVR on Sonnet would complement or substitute for scaling, remain unmeasured (budget). *Scaffold-config dependence.* PRVR's lift is conditional on the iteration and retry budgets we chose. Few long-iteration runs recovered the answer under either controller, and real deployments set similar limits; whether a larger budget would lift more cells or simply waste tokens on unrecoverable cases is an open question this benchmark does not settle. *Prompt-vocabulary overlap.* The Plan and Verify prompts share vocabulary with a minority of task families; the caveat-surfacing lift persists under their exclusion (disclosed in Sec. IV-A). *Scorer validation.* Caveat detection relies on deterministic keyword patterns over the prose answer fields; *List price vs billed cost.* The list-price methodology (Sec. IV-B3) is cache-invariant, but billed costs in real-world deployment depend on cache-reuse patterns that vary significantly.

D. Future work

Two directions follow directly from the channel attribution. Verify currently reads the submitted answer alongside a format check; grounding it in executed data probes (cheap assertions re-run over the produced files) would give the reactive half of the cycle evidence it presently lacks. Resolve, in turn, is a natural seat for calibrated escalation: measured as selective prediction, with escalation precision and recall scored against ground truth, flag-human would become a measurable diligence behavior rather than a budget stop. Separately, sub-task decomposition by a stronger LLM outside the scaffold let Flash-Lite produce valid output on four task families where its whole-task baseline fails to start, while self-decomposition by the weak model failed [9]; whether that configuration is cost-effective against running the stronger model directly remains open.

VI. Conclusion

DiscRun-Bench separates *discovering* latent hazards from *executing* on named hazards, and scores caveat surfacing separately from data correctness. Two findings emerged: hints help on two axes (attention and construction-path convergence), and one failure label hides distinct regimes, including a delivery-layer regime of acceptable answers lost at submission (Sec. III-E). A PRVR scaffold then showed the missing diligence is substantially elicitable: structuring the loop to ask lifts Haiku 4.5's caveat-surfacing rate from 0.74 to 0.90 and its mean task score by roughly 25% (both significant under paired tests), a gain carried by the preventive Plan orientation and delivery-layer robustness rather than the reactive retry, at list-price costs roughly $4\times$ below the larger-model reference it matches (Sec. IV-C). Silent failure in LLM data preparation is thus not only a question of what a model can notice but of what its operating loop asks it to say.

References

[1] M. Fan *et al.*, "AutoPrep: Natural language question-aware data preparation with a multi-agent framework," *Proceedings of the VLDB Endowment*, vol. 18, no. 10, pp. 3504–3517, 2025.

[2] P. Hosseini *et al.*, "Efficient solutions for an intriguing failure of LLMs: Long context window does not mean LLMs can analyze long sequences flawlessly," in *Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025)*, Abu Dhabi, UAE, 2025, pp. 1880–1891.

[3] F. Liu *et al.*, "Beyond functional correctness: Exploring hallucinations in LLM-generated code," *arXiv preprint arXiv:2404.00971*, 2024, accepted to IEEE Transactions on Software Engineering.

[4] M. Xiong *et al.*, "Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs," in *Proc. International Conference on Learning Representations (ICLR)*, 2024.

[5] Anthropic, "Introducing claude opus 4.8," https://www.anthropic.com/news/claude-opus-4-8, May 2026, accessed: 2026-05-31.

[6] Y. Lai *et al.*, "DS-1000: A natural and reliable benchmark for data science code generation," in *Proc. International Conference on Machine Learning (ICML)*, 2023.

[7] Y. Huang *et al.*, "DA-Code: Agent data science code generation benchmark for large language models," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[8] D. Zhang *et al.*, "DataSciBench: An LLM agent benchmark for data science," *arXiv preprint arXiv:2502.13897*, 2025.

[9] S. Guo *et al.*, "DS-Agent: Automated data science by empowering large language models with case-based reasoning," *Proc. International Conference on Machine Learning (ICML)*, 2024.

[10] S. Hong *et al.*, "Data interpreter: An LLM agent for data science," *arXiv preprint arXiv:2402.18679*, 2024.

[11] A. Egg *et al.*, "DABstep: Data agent benchmark for multi-step reasoning," *arXiv preprint arXiv:2506.23719*, 2025.

[12] Y. Lai *et al.*, "KramaBench: A benchmark for AI systems on data-to-insight pipelines over data lakes," *arXiv preprint arXiv:2506.06541*, 2025.

[13] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[14] X. Wang *et al.*, "Self-consistency improves chain of thought reasoning in language models," in *Proc. International Conference on Learning Representations (ICLR)*, 2023.

[15] N. Shinn *et al.*, "Reflexion: Language agents with verbal reinforcement learning," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[16] A. Madaan *et al.*, "Self-Refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.

[17] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 3–13, Dec. 2000.

[18] W. Zhou *et al.*, "Can LLMs clean up your mess? A survey of application-ready data preparation with LLMs," *IEEE Transactions on Knowledge and Data Engineering*, 2026, arXiv:2601.17058.

[19] OpenRefine Contributors, "OpenRefine: A free, open source tool for working with messy data, version 3.10.1," https://github.com/OpenRefine/OpenRefine/releases/tag/3.10.1, 2025.

[20] S. Kandel *et al.*, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, 2011, pp. 3363–3372.

[21] Great Expectations Contributors, "Great expectations: Tools for data validation, documentation, and profiling, version 1.17.2," https://pypi.org/project/great-expectations/1.17.2/, 2026.

[22] N. F. Liu *et al.*, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.