BEYOND EXECUTION: AUDITING EXPERIMENTAL FIDELITY IN LLM-DRIVEN SCIENTIFIC RESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are increasingly used to reproduce and extend scientific experiments, and their output is often judged by whether the generated code runs and reports plausible metrics. We argue that this criterion is insufficient and study *methodological hallucinations*: silent deviations from a reference experimental protocol that leave execution intact but undermine the scientific conclusion. Examples include reducing datasets or training budgets without reporting it, replacing a failed learning or generative component with a lookup or oracle function, and drawing conclusions at a scale where the method's claimed advantage cannot appear. From a case study of 30 long-horizon reproductions of machine learning papers under a single consumer-GPU budget, we derive a five-class taxonomy of such failures. 17 of the 30 runs (56.7%%) exhibit at least one, even though every run was produced by an agent that had the reference protocol in its prompt. To make such deviations easier to surface, we describe ABE-Ralph, a lightweight auditing protocol that encodes a paper's claims, required components, baselines, and metrics as a YAML contract, guides implementation through an 8-step workflow, and applies three inexpensive post-hoc checks: result coverage, LLM-based semantic review, and module presence. In a single-run comparison with four agent baselines, ABE-Ralph obtains the highest composite score (58.8, versus 51.0 for the strongest baseline) on a 21-task set whose per-system coverage differed (10–18 of 21 tasks). Because the systems also differ in backbone model and iteration budget, however, we cannot attribute this gap to auditing alone. Our findings suggest that evaluations of AI scientists should check whether an experiment faithfully tests the intended claim, not only whether it runs.

1 INTRODUCTION

Large Language Model (LLM) agents are expanding into autonomous research workflows, allowing systems to generate hypotheses, write code, run experiments, and draft reports (Lu et al., 2024; Liu et al., 2026; Boiko et al., 2023; Bran et al., 2024). Software engineering agents likewise resolve repository issues and produce working patches at high rates (Yang et al., 2024; Wang et al., 2025; Liu et al., 2024). These developments make it tempting to evaluate scientific agents by the same criterion used for software agents: whether the generated program executes and produces an output.

For scientific reproduction, that criterion is insufficient. A reproduction must satisfy four linked requirements: (1) the implementation contains the reference method described in the paper; (2) datasets, preprocessing, training, and baselines remain faithful to the original experiment; (3) the experiment is designed so that it can test the central claim under the available compute; and (4) the observed results constitute valid evidence for or against that claim. A script that returns exit code 0 and reports plausible metrics can fail all four.

The distinction matters most when agents face compute limits, missing dependencies, or failed checkpoints. When an experiment is hard to run, an agent may silently switch to a smaller dataset, fewer training steps, lower resolution, or random weights without reporting the change. It may replace a costly generative module with a lookup rule or oracle function that already contains the answer. It may also run the method at a scale too small for its claimed advantage to emerge and then conclude that the hypothesis is false. These actions preserve the appearance of progress while breaking the scientific logic. We call such deviations **methodological hallucinations**: silent, hard-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

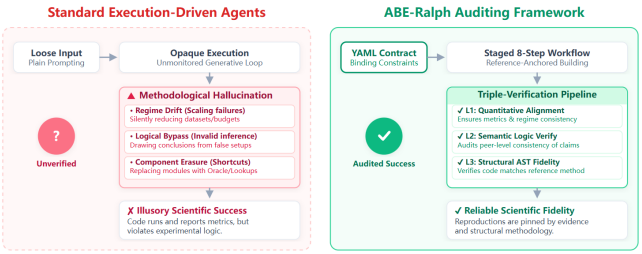

Figure 1: Execution-driven evaluation versus contract-based auditing. Under compute pressure, agents may take undeclared shortcuts (methodological hallucinations) that still yield a clean exit. ABE-Ralph records the reference protocol in a YAML contract and applies three lightweight post-hoc checks to the resulting code and outputs.

to-detect violations of the reference experimental protocol that preserve successful execution while undermining the validity of the conclusion (Abalo-Rodríguez & Pinheiro, 2025; Santhosh et al., 2026). The resulting metrics can lend support to false hypotheses or wrongly dismiss valid claims.

The problem is therefore not only that agents sometimes write incorrect code: they can produce a scientifically misleading experimental process out of technically valid artifacts. Execution-based evaluation cannot distinguish a faithful reproduction from a simplified implementation, an incomplete experiment, or one whose resource constraints invalidate its conclusion. Manual inspection, meanwhile, does not scale to long-horizon agent runs.

This paper makes the following contributions:

1. **Concept and taxonomy.** We define methodological hallucinations and propose a five-class taxonomy with explicit decision rules. The taxonomy separates hallucinations from genuine reproduction failures, in which the method is implemented faithfully but the reported result does not materialize (Section 4).
2. **Empirical case study.** We annotate 30 long-horizon reproduction runs spanning 12 machine learning domains. We report the frequency and co-occurrence of each failure class, analyze how resource constraints give rise to them, and show that the headline rate depends on how strictly "declared" is interpreted (Section 5.3).
3. **A lightweight auditing protocol.** We describe ABE-Ralph, which combines a YAML reference contract, an 8-step workflow, and three inexpensive post-hoc checks. We compare it with four agent baselines and state the confounds of this comparison explicitly (Sections 3 and 5.2). We release the pipeline code and the experiment contracts.

Scope. Our goal is to characterize a failure mode and to provide a practical audit aid, not a verifier with formal guarantees. We do not measure the precision or recall of the automated checks with a dedicated evaluation. We also do not claim that ABE-Ralph prevents methodological hallucinations: in the current implementation the auditor runs after the agent has finished and does not intervene, and the majority of the runs we annotate exhibit at least one hallucination. We discuss these limitations in Appendix I.

2 Related Work

LLM agents for scientific discovery. Autonomous research systems such as The AI Scientist (Lu et al., 2024), AutoResearchClaw (Liu et al., 2026), and Claw-AI-Lab (Wu et al., 2026) build end-to-end research workflows, but they focus primarily on novelty and successful execution and include no mechanism for auditing experimental fidelity. Domain-specific systems such as ChemCrow (Bran et al., 2024) and Coscientist (Boiko et al., 2023) automate laboratory tools without checking the methodological validity of the resulting experiments. Recent surveys (Wei et al., 2025) note that existing platforms tend to evaluate the surface form of scientific work rather than its internal validity. Our work focuses on this gap for reproduction tasks.

LLM agents for software engineering. Software engineering agents such as SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025) optimize for passing test suites under the SWE-bench paradigm (Jimenez et al., 2024; Yang et al., 2025; Deng et al., 2026), and code benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) measure functional correctness. In scientific tasks, however, passing tests or exiting cleanly is weak evidence, because an agent can bypass the core method with a trivial heuristic without raising any error. Unconstrained patches can also introduce subtle flaws (Sajadi et al., 2025).

Reproducibility benchmarks. Community efforts such as the ML Reproducibility Challenge and REPROLANG (Branco et al., 2020) rely on human reviewers to inspect replication reports, which does not scale to automated pipelines. Tools such as ReproZip (Chirigati et al., 2016) and Code Ocean capture computational environments but do not check whether the re-executed code implements the target method. Benchmarks for scientific agents, including ScienceAgentBench (Chen et al., 2025), MLE-bench (Chan et al., 2024), MLAgentBench (Huang et al., 2024), CORE-Bench (Siegel et al., 2024), and PaperBench (Starace et al., 2025), evaluate task completion, computational reproducibility, or rubric-based replication quality. PaperBench is the closest in spirit: it grades replications against paper-specific rubrics written by humans. Our work is complementary. Instead of proposing another success metric, we characterize *how* agents deviate from a reference protocol while still appearing to succeed, and we provide a contract format for recording and checking such deviations.

3 ABE-RALPH: A LIGHTWEIGHT CONTRACT-BASED AUDITING PROTOCOL

ABE-Ralph (Auto Baseline Experiment) wraps an LLM coding agent with three components: a reference contract that records what a faithful reproduction must contain (Section 3.2), an 8-step workflow in which the agent may refine its own implementation (Section 3.3), and an auditor that applies three checks once the agent has finished (Section 3.4). We first fix terminology.

3.1 PROBLEM FRAMING

We describe a reproduction task as a tuple $\mathcal{T} = \langle \mathcal{C}, \mathcal{D}, \mathcal{R} \rangle$:

- $\mathcal{C} = \{c_1, \ldots, c_n\}$ is a set of constraints extracted from the reference paper, specifying required components, datasets and protocol settings, baselines, metrics with their optimization direction, and the claims under test.
- $\mathcal{D}$ is the data and environment specification: the canonical datasets, preprocessing, and software environment.
- $\mathcal{R} = \{B_{comp}, B_{time}\}$ is the available resource budget (e.g., GPU memory and wall-clock time).

Given $\mathcal{T}$, an agent produces a program P and executes it on $\mathcal{D}$, yielding outputs $\mathcal{S}_E = P(\mathcal{D})$. Execution-driven evaluation counts the run as successful whenever

$$\text{Exit}(P(\mathcal{D})) = 0. \tag{1}$$

This condition says nothing about whether P satisfies $\mathcal{C}$. Under the pressure of $\mathcal{R}$, an agent may produce code that exits cleanly while violating some $c_k \in \mathcal{C}$, for instance by shrinking the dataset or dropping a required module. We call a run *faithful* if it satisfies every constraint in $\mathcal{C}$ within budget $\mathcal{R}$, or if it explicitly declares each deviation it makes. A methodological hallucination is an *undeclared* violation of $\mathcal{C}$ in a run that otherwise appears successful. ABE-Ralph records $\mathcal{C}$ explicitly, conditions the agent on it, and applies inexpensive checks that approximate parts of it.

3.2 REFERENCE CONTRACTS

The constraint set $\mathcal{C}$ is written as a declarative YAML contract (full example in Appendix A.4) with three groups of fields:

1. **Structural fields ($\mathcal{C}_{str}$)** name the components that the paper's method requires (e.g., specific layers, attention blocks, or loss functions), for example `critical_modules:`

Table 1: Which component makes which decision in ABE-Ralph.

Decision	Made by	Informed by
How to implement and scale the experiment	Agent	Contract (in prompt), workflow instructions
Whether to run another refinement iteration (max. 3)	Agent	Its own inspection of logs and results
Audit verdict (pass/flag)	Auditor, once at the end	L2 and L3 (Table 2)

`[UNetDecoder, SkipConnection]`. This field is optional; when it is absent, as in the DPR example, the module-presence check (L3) has nothing to check and passes by default. None of the 30 contracts defines `critical_modules`, so L3 was vacuous in all runs.

2. **Procedural fields ($\mathcal{C}_{proc}$)** specify datasets (including sample sizes), preprocessing, training regimes, and hyperparameter ranges. They make the intended scale explicit to the agent, so that downscaling is a visible departure rather than an implicit choice.
3. **Evaluative fields ($\mathcal{C}_{eval}$)** specify the primary metrics and their direction (e.g., `metric: F1, direction: maximize`), the baselines, and the paper's expected conclusions.

Who reads which field. Most fields are consumed only by the agent, as part of its prompt. The auditor reads the fields its checks use: `datasets` and `target_method` (L1), `key_claims`, `expected_paper_conclusions`, and `failure_conditions` (L2), and `critical_modules` (L3). In particular, no check compares the executed scale with the procedural fields; this is left to the agent and to manual review. The field-by-field mapping is given in Appendix A.4.

Contract provenance. Contracts are generated automatically from the paper by an LLM (GPT-4.1) and were not systematically verified by humans. We therefore call them *reference* contracts rather than ground truth. They can contain errors: the DPR contract, for example, states that in-batch negatives alone are more sample-efficient than hard negatives, whereas the original paper reports that adding BM25 hard negatives improves retrieval. We keep the contracts exactly as used in the experiments and discuss this issue in Appendix I.

3.3 Workflow and Refinement

The agent follows an 8-step workflow, from contract parsing and environment setup to implementation, execution, and reporting (Table 11 in the appendix). ABE-Ralph uses a commercial LLM coding agent invoked through the Claude Code CLI as its backbone; in our environment the API endpoint resolves to DeepSeek's `deepseek-v4-flash` model. After an initial implementation the agent may run up to three refinement iterations, followed by one ablation run. *Whether to refine is decided by the agent itself*, based on its own reading of logs and results; the auditor's checks (Section 3.4) do not trigger or block iterations. Table 1 summarizes which component makes which decision.

3.4 Three Lightweight Checks

After the agent has finished, the auditor runs three checks once on the final code and outputs. Table 2 summarizes what each check inspects, how it is implemented, and whether it contributes to the audit verdict. The exact procedures are given as algorithms in Appendix A.

L1: Result coverage (V_{cov}). Let $\mathcal{Q}_{req}$ be the (dataset, method) pairs whose results the contract requires, and let $\mathcal{J}$ be the result file produced by the run. L1 reports the fraction of required pairs present in the result file:

$$V_{cov} = \frac{\left|\{(d, m) \in \mathcal{Q}_{req} : \text{match}(d, m, \mathcal{J})\}\right|}{|\mathcal{Q}_{req}|}, \tag{2}$$

Table 2: The three checks as implemented. They run once, after the agent finishes. Only L2 and L3 contribute to the audit verdict; L1 is logged for inspection. The verdict is recorded but is not fed back to the agent.

Check	Inspects	Implementation	Verdict?
L1 Result coverage	Result JSON vs. required (dataset, method) pairs	Fuzzy key matching; no numerical comparison	No
L2 Semantic review	Key claims, expected conclusions, failure conditions, final report	Single LLM-judge call; binary decision parsed from output	Yes
L3 Module presence	Entry-point code vs. `critical_modules`	Name-level search for each listed module; passes if none listed	Yes

where match performs fuzzy matching on key names. L1 checks only whether results are present, not whether their values agree with the paper or preserve the paper's ranking of methods. It is logged but does not contribute to the verdict.

L2: Semantic review (V_{qual}). L2 calls an LLM judge (the backbone model) with the contract's `key_claims`, `expected_paper_conclusions`, and `failure_conditions`, together with the agent's final report; it does not see the code or the raw logs. The judge is asked whether the run faithfully implements the contracted protocol, and $V_{qual} \in \{0, 1\}$ is parsed from a `DECISION:` line in its response (Appendix A, Listing 1). Because the judge shares the backbone of the agent it evaluates, L2 may inherit that model's biases. We have not calibrated it against human judgments.

L3: Module presence (V_{struct}). For each module listed in `critical_modules`, L3 searches the run's entry-point code (`main.py`) for a matching name. It sets $V_{struct} = 1$ if every listed module is found, and also when no module is listed. L3 is a coarse check: it catches modules that were deleted or never implemented, but not modules that are present by name yet stubbed out or bypassed at runtime.

Audit verdict. The auditor flags a run unless both contributing checks pass:

$$\text{Pass} = V_{qual} \wedge V_{struct}. \tag{3}$$

The verdict is written to the run's audit record for a human reviewer. It does not stop the run, trigger a further iteration, or alter the agent's report. ABE-Ralph is therefore an *auditing* aid, not an enforcement mechanism; making the verdict feed back into the agent loop is a natural extension that we have not evaluated.

4 A Taxonomy of Methodological Hallucinations

We derived the taxonomy by reviewing the code, logs, and reports of the 30 reproduction runs described in Section 5.1 and grouping the recurring deviations. Table 3 gives each class with its decision rule; the per-run labels are in Table 12 and the case studies in Appendix F.

Boundary rules. Three rules keep the taxonomy from absorbing every unsuccessful run. (i) *Genuine reproduction failures are not hallucinations.* If the method is implemented and run faithfully and the reported effect simply does not appear (for example, because of training instability), the failure itself is not labelled M1; the run is marked as a genuine failure and receives only the labels its other behaviour warrants. (ii) *Declared deviations are not hallucinations.* A reduced budget or an unfinished experiment that is explicitly reported is a limitation of the run, not M2 or M5. (iii) *Changing the evaluation task or criterion* (e.g., evaluating on one task instead of the full suite, or replacing the paper's success criterion with a weaker one) is labelled M2 when undeclared. It is labelled M3 only when the report draws a conclusion about the claim from a scale at which the effect cannot appear.

Why these classes matter. M1 and M2 are the most dangerous for automated research because they yield plausible metrics while invalidating the experiment, and exit-code checks cannot see them.

Table 3: Taxonomy of methodological hallucinations. Counts are over $N = 30$ runs. A run may receive several labels; 13 runs received none (*Clean*).

ID	Class	Decision rule (label if . . .)	n (%)
M1	Method integrity collapse	a required component is removed, stubbed, or replaced by a lookup, oracle, or trivial heuristic, while the run still reports results for the method	3 (10.0%)
M2	Silent protocol degradation	datasets, evaluation tasks, evaluation criteria, training budget, resolution, or other contracted settings are reduced or substituted *without being declared* in the report	8 (26.7%)
M3	Scale-driven conclusion inversion	the run is executed at a scale where the claimed effect cannot be expected to appear, and the report nonetheless draws a conclusion about the claim	3 (10.0%)
M4	Quantitative key mismatch	reported results cannot be aligned with the contracted metrics or methods (e.g., renamed keys), so the claimed comparison is not actually evidenced	2 (6.7%)
M5	Incomplete execution	the run stops (OOM, timeout, unrecovered crash) before producing all contracted results, while the output does not make this incompleteness explicit	15 (50.0%)

M3 reflects the fact that a method's advantage often depends on the compute regime, so scale limits can silently invert a conclusion. M4 is closer to a reporting failure than a methodological one. We keep it because it silently breaks automatic comparison with the paper. M5 is the most frequent class. It is *not* intrinsically deceptive, but it becomes a hallucination when the incomplete output is presented as a finished reproduction. Each class suggests a corresponding check. M1 motivates structural checks (L3, which is only a coarse approximation), M1 and M2 motivate semantic review against the contract (L2), M4 motivates schema normalization, and M3 and M5 motivate tracking the scale actually executed against the contracted scale, which ABE-Ralph does not yet do.

5 Experiments

We address three questions. **RQ1**: How do agent systems compare on the reproduction tasks when all of them receive the contract content? **RQ2**: How frequent are the classes of methodological hallucination, and how do they co-occur? **RQ3**: How does the removal of each check's instructions affect the composite score?

5.1 Experimental Setup

Tasks. We construct 30 reproduction tasks, each based on a published machine learning paper. The tasks span natural language processing, computer vision, reinforcement learning, graph learning, and time-series modelling, and we group them into 12 domains (task list in Appendix D, Table 13). Each task has a reference contract (Section 3.2).

Compute budget. All runs use a single laptop GPU (NVIDIA RTX 4070 Laptop, 8 GB VRAM), with per-run time budgets drawn from the reference contracts (median 26 h, range 10–58 h). Several original experiments (e.g., large-scale ViT pre-training, ImageNet-scale training, MuJoCo control) cannot be reproduced at their original scale under this budget. We keep this setting deliberately. Resource-constrained reproduction is realistic for many practitioners, and it is exactly the situation in which agents are tempted to downscale silently. The setting does, however, inflate the frequency of M3 and M5, and we interpret those classes as describing agent behaviour *under constraints*, not intrinsic agent failures.

Systems. We compare five systems:

- **Raw LLM**: the same commercial LLM API family as the other systems (resolved to DeepSeek's `deepseek-v4-flash` in this environment), invoked single-shot with no tool loop.

- **ARC** (Autonomous Research Catalyst) (Liu et al., 2026): a sequential framework that follows a fixed step-by-step reproduction recipe.
- **Claw-AI-Lab** (Wu et al., 2026): a template-driven research agent.
- **Claude Code CLI**: the interactive coding agent that serves as ABE-Ralph's backbone, run without the contract workflow.
- **ABE-Ralph**: Claude Code CLI wrapped with the contract, the 8-step workflow, up to three agent-initiated refinement iterations, and the post-hoc auditor (Section 3).

Each baseline receives the task's contract: the raw-LLM and Claude Code CLI baselines embed the full contract text in their prompt (the Claude Code CLI adapter truncates very long plans at 8000 characters), while ARC and Claw-AI-Lab extract the task topic and method name from the plan into their own configuration with the contract file placed in the workspace. ABE-Ralph additionally re-exposes the agent to the contract at every workflow step. The systems are *not* matched in backbone model, number of iterations, tool calls, tokens, or wall-clock time, and each system was run once per covered task, with coverage differing across systems (Section 5.2).

Scoring. The scoring pipeline rates each run on six dimensions (Table 9 in Appendix B). Five of them are computed by the pipeline from the run's code, logs, and outputs, without an LLM judge, and four of them form the composite score $S \in [0, 100]$:

$$S = 0.20\, S_A + 0.25\, S_B + 0.25\, S_C + 0.30\, S_D, \tag{4}$$

where S_A scores repository structure and design, S_B reliability (run-time exceptions and crashes), S_C the rigor of the evaluation protocol, and S_D completeness against the contract. The weights were set heuristically, placing slightly more weight on completeness. The remaining two dimensions, E (alignment with the contract) and F (overall report quality), are excluded from S and reported as diagnostics: E is rule-based (module existence, blueprint coverage, and condition coverage against the contract), while F is LLM-judged. An LLM-judged alignment score could reward a report for echoing the contract's wording, which ABE-Ralph's workflow encourages. The same pipeline is applied to all systems. We treat S as a coarse summary of engineering completeness and protocol quality. It does not measure methodological fidelity, which we analyze separately through the taxonomy (Section 5.3).

Annotation. The taxonomy labels were assigned by an LLM-based annotation agent following the decision and boundary rules in Section 4, and were then verified by the authors. They were *not* derived from the auditor's outputs. All 30 labelled runs are ABE-Ralph runs, i.e., the same runs that enter the ABE-Ralph column of RQ1; runs of the other systems were scored but not annotated. Labels were assigned in a single pass and we have not measured inter-annotator agreement.

5.2 Overall Comparison (RQ1)

Figure 2 reports the composite scores. The comparison ran on 21 tasks (5 reproduction tasks and 16 NatureBench tasks); per-system coverage was 10–18 of the 21 (10 for ABE-Ralph, 18 for ARC, Claw-AI-Lab and Claude Code CLI, 15 for Raw LLM), so each score averages over a somewhat different subset. ABE-Ralph scores 58.8, followed by Claude Code CLI (51.0), Claw-AI-Lab (39.9), ARC (33.0), and Raw LLM (30.8). These are single-run results without variance estimates, so small differences should not be over-interpreted.

The most informative comparison is between ABE-Ralph and Claude Code CLI, because ABE-Ralph wraps that same CLI as its backbone. The 7.8-point gap reflects the combined effect of the contract-driven workflow and the additional refinement iterations; since the auditor does not feed back into the run, it cannot have contributed to the score directly. Our data do not separate the remaining factors. According to the per-dimension breakdown (Appendix B), Claude Code CLI's score comes mainly from design and reliability. Because we annotated only ABE-Ralph runs, we cannot say how often the baselines hallucinate; a high composite score should not be read as evidence of fidelity for any system. The gap between ABE-Ralph and Raw LLM is larger but less informative.

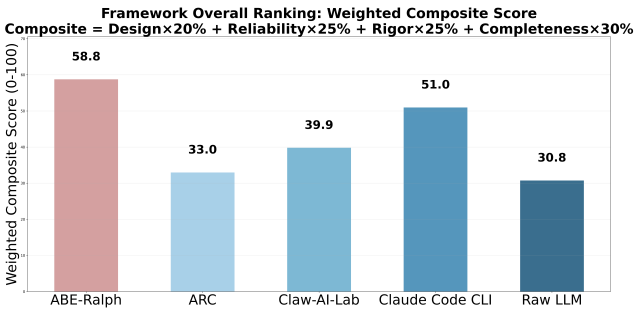

Figure 2: Composite score S (dimensions A–D) per system on the 21-task comparison set. Per-system coverage was 10–18 of 21 tasks (ABE-Ralph 10, ARC/Claw-AI-Lab/Claude Code CLI 18, Raw LLM 15); the figure does not annotate the per-bar N.

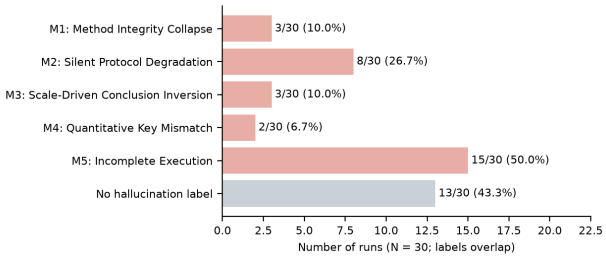

Figure 3: Share of ABE-Ralph runs ($N = 30$) that received each label. Labels are not mutually exclusive, so percentages do not sum to 100. *Clean* runs received no label. Generated from the label file by `make_taxonomy_assets.py`.

5.3 Distribution of Methodological Hallucinations (RQ2)

Figure 3 and Table 3 show the distribution of labels over the 30 ABE-Ralph runs. 13 runs (43.3%%) are **Clean**. The remaining 17 (56.7%%) exhibit at least one methodological hallucination.

Contract-based guidance does not prevent the problem. Every annotated run was produced by an agent that had the contract in its prompt and was repeatedly pointed back to it by the workflow. Most runs still deviate from the protocol. Because the auditor only runs after the fact, this finding concerns prompt-level guidance: telling the agent what a faithful reproduction requires is not enough to make it produce one. We see this as the main empirical finding of the paper, and it motivates treating fidelity as a first-class evaluation target, checked by something other than the agent itself.

Resource limits drive the most common classes. M5 (Incomplete Execution) is the most frequent class (15 runs, 50.0%%), largely because runs hit memory or time limits. M2 (Silent Protocol Degradation) follows (8 runs, 26.7%%): agents reduce data, epochs, or evaluation scope, or substitute a weaker evaluation criterion, without reporting the change. Taken together, these reflect a trade-off that agents face at the compute ceiling: stop early (M5), or downgrade silently (M2). M1 (3 runs), M3 (3), and M4 (2) are less frequent. M1 is nonetheless the most consequential, because its metrics look plausible.

The headline rate depends on what counts as "declared". 5 of the 17 affected runs carry only an M5 label. Whether an unfinished run is a hallucination hinges on whether its report makes the incompleteness explicit, which involves judgment. Under the strictest reading, in which all 5 of these runs count as declared, the share of affected runs falls from 56.7%% to 40.0%% (12 of 30). We therefore report the range rather than a single figure. The runs affected by M1–M4 do not depend on this judgment.

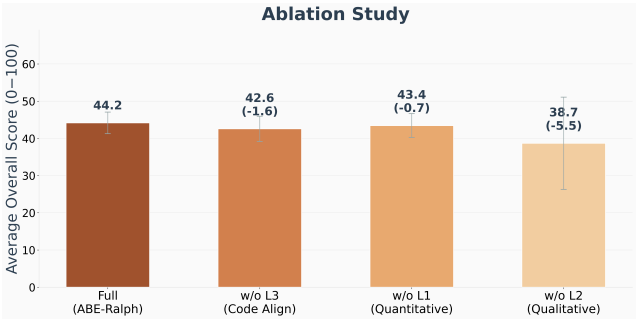

Figure 4: Instruction ablation. Each variant removes the prompt instructions associated with one check. Scores are comparable only within this figure, not with Figure 2.

Failures co-occur. The 17 affected runs carry 31 labels in total, so deviations tend to occur together. In the SimCLR run (M1 and M2), for example, the agent replaced the paper's evaluation criterion with a much weaker "better than chance" criterion and did not declare the substitution, and it queued but never ran the baseline encoders, so the core comparison is absent (Appendix F). Most other multi-label runs combine M5 with one further class; per-run labels are listed in Table 12.

5.4 Instruction Ablation (RQ3)

We ablate ABE-Ralph by removing, one at a time, the workflow instructions associated with each check (Appendix A, Table 6). This removes the *prompt-level instructions* only. The auditor itself runs after the agent in every variant and does not influence the run, so the ablation measures how the instructions change the agent's behaviour. Figure 4 reports the result.

Scale of the scores. The ablation and RQ1 use the same scoring formula and weights over dimensions A–D; the numbers differ because they average over different task sets — the ablation ran on the 30 reproduction tasks, while RQ1 ran on the 21-task comparison set (Section 5.2). The full configuration therefore scores 44.2 here, versus 58.8 in RQ1. We compare each variant only with the full configuration of the same task set.

Removing the L1 instructions changes the score by -0.7 and removing the L3 instructions by -1.6. Given single runs, both differences fall within plausible run-to-run variation. Since L1 does not contribute to the verdict in our implementation (Table 2), the L1 result should not be read as evidence about the value of numerical checking. Removing the L2 instructions produces the largest drop (-5.5) and larger variance across tasks. This *suggests* that instructing the agent to review its work against the contract's claims carries most of the benefit, but confirming it would require repeated runs.

6 Conclusion

We studied methodological hallucinations: undeclared deviations from a reference protocol that leave an agent's experiment runnable but scientifically misleading. Across 30 constrained reproduction runs, 17 (56.7%%; 40.0%% under the strictest reading) exhibited at least one such deviation, even though the agent had the reference protocol in its prompt. Incomplete execution and silent protocol degradation were the most common, and failures tended to co-occur. We proposed a five-class taxonomy with explicit decision rules and described ABE-Ralph, a lightweight contract-and-check protocol that obtained the highest composite score among the systems we compared, subject to the confounds stated in Appendix I. Feeding audit verdicts back into the agent loop, measuring detection accuracy against human labels, controlling for backbone and budget, and verifying reference contracts are the most important next steps. More broadly, our results suggest that evaluations of AI scientists should ask whether an experiment faithfully tests the intended claim, not only whether it runs.

AI USE STATEMENT

We used generative AI tools in three ways. (1) LLM coding assistants helped write and debug the ABE-Ralph framework and the benchmark pipeline. (2) The reference contracts were generated by an LLM (GPT-4.1) from the source papers, as described in Section 3.2, and were used without systematic human verification. (3) We used LLMs for language polishing of parts of the manuscript. In addition, the systems under study—LLM-based coding agents invoked through commercial APIs, which in this environment resolved to DeepSeek's `deepseek-v4-flash` model—produced all experimental runs; an LLM judge performs the auditor's semantic review (L2), and one diagnostic dimension (F) is LLM-judged, while the composite score is computed by the rule-based parts of the pipeline. The taxonomy labels were assigned by an LLM-based annotation agent and then verified by the authors. All reported numbers are computed from the logged outputs of these runs. The authors reviewed all AI-assisted code and text and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies the automated reproduction and auditing of machine learning experiments. It involves no human subjects and no personal data. All tasks reproduce published methods on publicly available datasets. The failure taxonomy documents limitations of current AI-assisted research practice rather than harmful capabilities. A possible risk is that audit tools create unwarranted trust in agent outputs. We therefore state the limitations of our checks explicitly and do not present them as guarantees.

REPRODUCIBILITY STATEMENT

The algorithms of ABE-Ralph, the YAML contract schema with a worked example, and per-run case studies for all 30 runs are provided in Appendices A, A.4, and F. The per-run labels are kept in a single file from which all taxonomy tables, counts, and Figure 3 are generated. All datasets are public. The systems depend on commercial LLM APIs (resolved to DeepSeek's `deepseek-v4-flash` in this environment; the contracts were generated by GPT-4.1), so exact re-runs may be affected by model updates. We will release the pipeline code and the experiment contracts anonymously with the supplementary material.

REFERENCES

Inés Abalo-Rodríguez and Ana P. Pinheiro. The hitchhiker's guide to hallucination research. *Consciousness and Cognition*, 136:103941, 2025. ISSN 1053-8100. doi: 10.1016/j.concog.2025.103941. URL https://www.sciencedirect.com/science/article/pii/S1053810025001345.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Daniil A. Boiko, Robert MacKnight, Benjamin Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0. URL https://doi.org/10.1038/s41586-023-06792-0.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024.

António Branco, Nicoletta Calzolari, Piek Vossen, Gertjan Van Noord, Dieter van Uytvanck, João Silva, Luís Gomes, André Moreira, and Willem Elbers. A shared task of a new, collaborative type to foster reproducibility: A first exercise in the area of language science and technology with REPROLANG2020. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 5539–5545, Marseille, France, may 2020. European Language Resources Association. URL https://aclanthology.org/2020.lrec-1.680/.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering, 2024. URL https://arxiv.org/abs/2410.07095.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6z4YKr0GK6.

Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. Reprozip: Computational reproducibility with ease. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 2085–2088, New York, NY, USA, 2016. Association for Computing Machinery. doi: 10.1145/2882903.2899401. URL https://doi.org/10.1145/2882903.2899401.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa R Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=uEVTdoAbnK.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning (ICML)*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Meng Chen, Congyu Zhang, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Lu Feng, Xujiang Zhao, Haifeng Chen, Jiawei Zhou, Xiao Wang, Weitong Zhang, Hongtu Zhu, Yun Li, Jieru Mei, Hongliang Fei, Jiaheng Zhang, Linjie Li, Linjun Zhang, Yuyin Zhou, Sheng Wang, Caiming Xiong, James Zou, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. AutoResearchClaw: Self-reinforcing autonomous research with human-AI collaboration, 2026. URL https://arxiv.org/abs/2605.20025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, pp. 52989–53046, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9df36b21ff4ee211a8b71ee8b7e9f57-Paper-Conference.pdf.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. arXiv preprint arXiv:2408.06292, 2024.

Amirali Sajadi, Kostadin Damevski, and Preetha Chatterjee. How safe are AI-generated patches? A large-scale study on security risks in LLM and agentic automated program repair on SWE-bench, 2025. URL https://arxiv.org/abs/2507.02976.

Varkey Nadakkavukaran Santhosh, Rhea Vas, Biswadeep Roychowdhury, Karthiga Sakthi, and Margubur Rahaman. Development and content validation of the CAREFUL-AI framework for evaluating AI-generated scientific manuscripts: an exploratory cross-platform study. *Research Evaluation*, 35:rvag024, 01 2026. ISSN 0958-2029. doi: 10.1093/reseval/rvag024. URL https://doi.org/10.1093/reseval/rvag024.

Zachary S. Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebl, and Arvind Narayanan. CORE-Bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024.

Giulio Starace et al. PaperBench: Evaluating AI's ability to replicate AI research. In *International Conference on Machine Learning (ICML)*, 2025.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, Kai Tian, Zhenzhao Yuan, Jincheng Zhong, Weizhi Wang, Ning Ding, Bowen Zhou, and Kaiyan Zhang. NatureBench: Can coding agents match the published SOTA of Nature-family papers?, 2026. URL https://arxiv.org/abs/2606.24530.

Jiaqi Wei, Yuejin Yang, Xiang Zhang, Yuhan Chen, Xiang Zhuang, Zhangyang Gao, Dongzhan Zhou, Guangshuai Wang, Zhiqiang Gao, Juntai Cao, Zijie Qiu, Ming Hu, Chenglong Ma, Shixiang Tang, Junjun He, Chunfeng Song, Xuming He, Qiang Zhang, Chenyu You, Shuangjia Zheng, Ning Ding, Wanli Ouyang, Nanqing Dong, Yu Cheng, Siqi Sun, Lei Bai, and Bowen Zhou. From AI for science to agentic science: A survey on autonomous scientific discovery, 2025. URL https://arxiv.org/abs/2508.14111.

Fan Wu, Cheng Chen, Zhenshan Tan, Taiyu Zhang, Xinzhen Xu, Yanyu Qian, Dingcheng Gao, Lanyun Zhu, Qi Zhu, Yi Tan, Deyi Ji, Guosheng Lin, Tianrun Chen, Deheng Ye, and Fayao Liu. Claw AI lab: An autonomous multi-agent research team, 2026. URL https://arxiv.org/abs/2605.22662.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alex Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *International Conference on Learning Representations*, pp. 2794–2829, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/07d6332ae36730707fddddba736d7b6c-Paper-Conference.pdf.

Jiakang Yuan, Xiangchao Yan, Bo Zhang, Tao Chen, Botian Shi, Wanli Ouyang, Yu Qiao, Lei Bai, and Bowen Zhou. Dolphin: Moving towards closed-loop auto-research through thinking, practice, and feedback. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21768–21789, Vienna, Austria, jul 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1056. URL https://aclanthology.org/2025.acl-long.1056/.

A IMPLEMENTATION DETAILS OF ABE-RALPH

This section documents what the current ABE-Ralph prototype actually does. Where the implementation is simpler than one might expect from the names of its components, we say so explicitly.

A.1 OVERVIEW: AN AGENT LOOP PLUS A POST-HOC AUDITOR

ABE-Ralph has two components that share a YAML reference contract (Appendix A.4):

1. **The reproduction agent** (`ralph_github.py`): a staged, prompt-driven workflow that drives Claude through intent analysis, dataset search, implementation, execution, reporting, a bounded improvement loop and an ablation run.
2. **The post-hoc auditor** (`verify_reproduction.py`): three lightweight checks, namely a result-coverage check (L1), an LLM-judge review (L2) and a module-presence check (L3), run on the final workspace.

Table 4 summarizes which component makes which decision. The most important point is that **the auditor does not gate the agent loop**. Whether to run another improvement iteration is decided by the agent itself. The auditor verdict is computed afterwards and is what we report.

Table 4: Where decisions are made in ABE-Ralph.

Decision	Made by	Mechanism
Continue or stop improving	Agent (LLM)	The analysis step emits `HYPOTHESES_SUPPORTED` or `OPTIMIZATION_DONE`; at most `max_retries`$=3$ iterations (Alg. 2).
Recovery from errors (e.g., OOM)	Agent (LLM)	Self-healing: the agent inspects the error, patches its own code and re-runs; no programmatic restriction on what may be changed.
Dataset authenticity	Agent + string check	The dataset step emits a warning if `DATA_READY` is absent, and the run always continues; there is no abort flag in the current implementation.
Final audit verdict	Auditor	L2 decision, downgraded if `metrics.json` is missing, overridden to `FAILED` if L3 fails (Alg. 8). L1 is logged only.
Hallucination labels (M1–M5)	LLM-based annotation agent (verified by authors)	Rubric in Appendix D; regex tags are used only as hints.

A.2 THE REPRODUCTION AGENT (`RALPH_GITHUB.PY`)

Initialization and engineering details. The agent parses the contract, detects reproduction vs. discovery mode, maps the contract's `time_budget` to per-step timeouts, and resumes from a checkpoint if one exists. Interrupts are handled with a standard self-pipe pattern: a signal thread writes to a pipe that the main thread `selects` on together with the subprocess stdout. This keeps long subprocesses interruptible. These details affect robustness but not the methodology.

Staged workflow. Algorithm 1 lists the stages. Every stage is a Claude call with a stage-specific prompt and timeout.

Reproduction vs. discovery mode. Table 5 contrasts the two modes. All 30 runs analysed in this paper use reproduction mode.

Improvement cycle. Algorithm 2 shows one iteration. The agent audits its own report, patches and re-runs the code, and then decides in a separate analysis call whether to stop. This self-audit is prompt-driven. It is distinct from the post-hoc auditor in §A.3, which does not run inside this loop.

Algorithm 1 Staged workflow of the reproduction agent

procedure RUNWORKFLOW()
RUNCLAUDESTEP(prompt_intent, timeout=300s)
RUNCLAUDESTEP(prompt_dataset, timeout=3600s)
{the dataset step warns if DATA_READY is absent; the run always continues}
RUNCLAUDESTEP(prompt_research, timeout=1800s)
RUNCLAUDESTEP(prompt_blueprint, timeout=300s)
RUNCLAUDESTEP(prompt_smoke_test, timeout=1800s)
RUNCLAUDESTEP(prompt_full_execution, timeout=3600s)
RUNCLAUDESTEP(prompt_synthesis, timeout=600s)
RUNCLAUDESTEP(prompt_skills)
for $i = 1$ **to** max_retries **do**
  out $\leftarrow$ RUNIMPROVEMENTCYCLE(i)
  if HYPOTHESES_SUPPORTED $\in$ out $\vee$ OPTIMIZATION_DONE $\in$ out **then**
    break {stopping is decided by the agent, not by the auditor}
  end if
end for
RUNABLATIONSTUDY()
end procedure

Table 5: Comparison of reproduction and discovery modes.

Step	Reproduction Mode	Discovery Mode
1. Intent	Analyze YAML contract	Read problem/README.md
1.5. Dataset	Search HF/Kaggle/OpenML	Analyze local files and evaluator.py
2. Research	Search reference code repos	Search domain literature
3. Blueprint	Build main.py from reference	Design solution from scratch
4b. Execute	Train & output metrics.json	Train & execute evaluator.py
5. Synthesis	Check YAML criteria	Summarize evaluator score

Improvement-cycle usage. The refinement loop is capped at max_retries $= 3$; the agent stops earlier whenever its own analysis emits HYPOTHESES_SUPPORTED or OPTIMIZATION_DONE (Algorithm 2). The number of iterations actually used in each run is recorded per run in experiment_state.json (see the example below); the aggregated distribution over the 30 runs is not reported in this revision.

Checkpoint state. Execution state is persisted in experiment_state.json. The listing below reproduces a real state file verbatim (run bm_004_bert, ABE-Ralph workspace); the output_preview of the completed step is truncated for space. All other steps are still pending at the time this snapshot was written.


```
{
  "plan_path": "experiment_plan.yaml",
  "plan_mtime": 1782380188.5583858,
  "started_at": "2026-06-25T17:36:28.672910",
  "last_updated": "2026-06-25T17:37:18.693856",
  "steps": {
    "intent_discovery": {
      "status": "completed",
      "output_preview": "Now I have all the data. ... [truncated]",
      "completed_at": "2026-06-25T17:37:18.693784",
      "timeout_used": 300,
      "duration_seconds": 37.3,
      "iteration": 0
    },
    "dataset_search": {"status": "pending"},
    "research_step": {"status": "pending"},
    "blueprint": {"status": "pending"},
    "smoke_test": {"status": "pending"},
    "full_execution": {"status": "pending"},
    "synthesis_reporting": {"status": "pending"},
    "skill_extraction": {"status": "pending"}
  },
  "iteration": {
    "current": 0,
```

Algorithm 2 One iteration of the agent's improvement cycle

procedure RUNIMPROVEMENTCYCLE*(iteration)*
`prompt_imp` ← ConstructPrompt({
  Audit: "Analyze experiment_result.md and raw data",
  CodeCheck: "Verify main.py against YAML contract",
  Fix: "Patch code and re-execute main.py"})
RUNCLAUDESTEP(`prompt_imp`, step_name="imp_v" + *iteration*)
`prompt_ana` ← ConstructPrompt({
  Review: "Compare results against YAML success criteria",
  Signal: "Output HYPOTHESES_SUPPORTED or OPTIMIZATION_DONE"})
`out` ← RUNCLAUDESTEP(`prompt_ana`, step_name="ana_v" + *iteration*)
TRACKITERATION(*iteration*, `HYPOTHESES_SUPPORTED` ∈ `out`, `OPTIMIZATION_DONE` ∈ `out`)
return `out`
end procedure

```
    "max_retries": 3,
    "convergence_threshold": 0.01,
    "history": []
  }
}
```

Ablation procedure. Algorithm 3 and Table 6 describe the ablation. **The ablation removes prompt directives from the agent. It does not disable auditor checks.** The configuration names in the main text (w/o L1/L2/L3) refer to the prompt directives associated with each auditing concept, not to the auditor modules L1–L3 in §A.3. To avoid confusion, Table 6 gives each configuration a descriptive name. Each ablation configuration re-runs the full agent pipeline in an isolated workspace: PREPAREISOLATEDWORKSPACE copies the original workspace, symlinks the data directory and the virtual environment, and copies the (possibly prompt-modified) plan into it; the verifier and SCOREWORKSPACEMULTIDIM then run on the freshly produced workspace (Algorithm 3).

Algorithm 3 Ablation procedure

procedure RUNABLATIONSTUDY*()*
for all `cfg` ∈ {Full, NoL3, NoL1, NoL2} **do**
  `ws` ← PREPAREISOLATEDWORKSPACE(`cfg`)
  SWITCHTO(`ws`)
  if `cfg` == NoL3 **then**
    REMOVEREFERENCECODECONSTRAINTS()
  else if `cfg` == NoL1 **then**
    REMOVEQUANTITATIVETARGETPROMPTS()
  else if `cfg` == NoL2 **then**
    DISABLEDEEPAUDITPROMPTS()
  end if
  `decision` ← EXECUTEEVALUATION()
  `scores` ← SCOREWORKSPACEMULTIDIM(`cfg`)
  RESTOREWORKSPACE()
end for
end procedure

Table 6: Prompt-level ablation mapping. Only agent prompts are modified; the post-hoc auditor is unchanged.

Config (main text)	Descriptive name	Removed prompt directives
w/o L3	w/o reference-code guidance	Skip repository search; remove code-guidance instructions
w/o L1	w/o metric-target prompts	Omit comparison against YAML metric success criteria
w/o L2	w/o self-audit prompts	Remove deep-audit and root-cause prompt blocks

Scoring. Ablation workspaces are scored with SCOREWORKSPACEMULTIDIM (Algorithm 4). Its composite uses only dimensions A–D; the dimensions are defined in Appendix B.

Algorithm 4 Composite workspace score used in the ablation

1: **procedure** SCOREWORKSPACEMULTIDIM*(label)*
2: `s` ← EXPERIMENTINTEGRATOR(`workspace`).COMPUTEMULTIDIMSCORE()
3: **return** $S_{\text{comp}} = 0.20\, s_A + 0.25\, s_B + 0.25\, s_C + 0.30\, s_D$ {E, F not included}
4: **end procedure**

A.3 THE POST-HOC AUDITOR (`VERIFY_REPRODUCTION.PY`)

The auditor loads the contract, locates the latest report, and extracts the datasets, baselines and target method. It then runs the three checks below. We use the same names as Table 1 of the main text.

L1: Result coverage check (logged only). L1 checks that `metrics.json` has an entry for every contract dataset and, under it, an entry for the target method. Keys are matched by fuzzy string matching (Algorithm 5). **L1 does not compare numeric values, and its result does not affect the final verdict** (Algorithm 8, line 3). The contract fields `expected_metric_values`, `baselines` and `success_conditions` are *not* read by L1. They reach the agent only through its prompts.

Algorithm 5 L1: result coverage check

1: **procedure** RUNDYNAMICQUANTITATIVECHECK*(metrics_data)*
2: **if** *metrics_data* is NULL **then**
3: **return** (FALSE, "metrics.json missing")
4: **end if**
5: **for all** `ds` ∈ `self.datasets` **do**
6: `k_ds` ← FUZZYMATCH(`ds`, *metrics_data.keys()*)
7: **if** `k_ds` is NULL **then**
8: **return** (FALSE, "Dataset missing")
9: **end if**
10: `k_tgt` ← FUZZYMATCH(`self.target`, *metrics_data*[`k_ds`])
11: **if** `k_tgt` is NULL **then**
12: **return** (FALSE, "Target method missing")
13: **end if**
14: **end for**
15: **return** (TRUE, "All required entries present")
16: **end procedure**

L2: LLM-judge review. L2 is a single Claude call that plays the role of a meta-reviewer. It receives the contract's `key_claims` and the agent's final report, and its output is parsed with a regular expression into `MET_ALL`, `MET_PARTIALLY` or `FAILED` (Algorithm 6). L2 sees the report, not the code or raw logs. It can therefore be misled by a report that overstates what was run, and it shares the biases of the model family used by the agent. The full prompt is given in Listing 1.

Listing 1: Full L2 judge prompt from `verify_reproduction.py`, translated from the original Chinese. Fields marked { . . . } are filled from the contract and the run.

```
You are a senior, rigorous academic paper review expert (Meta-Reviewer),
responsible for auditing the scientific soundness and conclusion
consistency of a reproduction experiment.

[Experiment Background]
- Research Topic (Topic): {topic}
- Original Paper (Paper): {paper_title}
- Evaluation Datasets (Datasets): {datasets}
- Target Method to Reproduce (Target Method): {target_method}
- Baseline Methods for Comparison (Baselines): {baselines}
- Core Evaluation Metrics (Metrics): {metrics}

```

Algorithm 6 L2: LLM-judge review

procedure RUNUNIVERSALQUALITATIVECHECK*()*
`prompt` ← ConstructPrompt({
  Role: "Senior meta-reviewer auditing reproduction",
  Claims: `self.plan.key_claims`,
  Report: READFILE(`self.report_path`)})
`out` ← INVOKECLAUDE(`prompt`)
`decision` ← PARSEDECISION(`out`) {Regex: DECISION:\s*(MET_ALL—MET_PARTIALLY—FAILED)}
return `decision`
end procedure

```
[Key Claims of the Original Paper (Key Claims)]:
{key_claims}

[Expected Conclusions of the Original Paper (Expected Conclusions)]:
{expected_paper_conclusions}

[Preset Reproduction Failure Red Lines (Failure Conditions)]:
{failure_conditions}

[Content of the Actual Reproduction Experiment Report]:
"""
{report_content}
"""

Based on the above context, objectively review the reproduction report
and write a review:

1. **Quantitative alignment analysis**: Do the specific numerical trends
   in the reproduction report agree in direction (relative magnitudes,
   improvement margins) with the key claims and expected conclusions of
   the original paper?
2. **Qualitative logic analysis**: Does the experiment report verify the
   non-numerical hypotheses raised by the original paper (e.g., the design
   of an ablation study, the necessity of a certain architecture or
   strategy)? Are the conclusions consistent?
3. **Red lines and risk screening**: Does the report trigger any preset
   reproduction failure red line (Failure Conditions)? Are there signs of
   an invalid reproduction caused by implementation issues, data leakage,
   or reduced scale?
4. **Honesty assessment**: For reduced-scale reproductions, does the
   report honestly discuss the differences in absolute values and give
   reasonable scientific explanations?

[Output Format Specification]
You must output the final verdict decision as the very first line of
your answer, in the format:
DECISION: [MET_ALL] (fully consistent / reproduction successful)
or DECISION: [MET_PARTIALLY] (basically consistent, but with flaws or
some claims not holding)
or DECISION: [FAILED] (not consistent / experiment failed / red line
triggered)

Then, provide structured, detailed review comments.
```

L2 invocation details. The prompt above is passed to a single `claude` CLI call; the exact subprocess command is `claude --permission-mode bypassPermissions <prompt>`, at the CLI's default model and decoding settings (no `--model` or `--temperature` override), with a 1800 s timeout. The decision is extracted with the regular expression `DECISION:\s*[\[]?(MET_ALL|MET_PARTIALLY|FAILED)[\]]?`; if the response contains no such line, the decision is recorded as `UNKNOWN`. If `MET_ALL` is returned while `metrics.json` is missing, the verdict is downgraded to `MET_PARTIALLY` (Algorithm 8).

L3: Module presence check. L3 reads `main.py` and checks that every name listed in `target_method.critical_modules` appears as a symbol in its AST (Algorithm 7). This is a name-level presence check. It does not verify that the module is called, trained or used correctly. `critical_modules` is an optional field. **When it is absent, as in**

the DPR contract in Listing 2, the loop is empty and L3 passes vacuously. None of the 30 reproduction contracts defines `critical_modules`, so L3 passed vacuously in all analysed runs; the field appears only in discovery-mode plans. The underlying `alignment_verifier.py` (`CodeAlignmentVerifier`) implements the module-existence search that supplies `overall_pass`, and additionally reports *advisory* blueprint-coverage and structural-fidelity checks. Only `overall_pass` (module existence) affects the verdict; the advisory results surface only as warnings (Algorithm 8).

Algorithm 7 L3: module presence check

procedure RUNCODEALIGNMENTCHECK*()*
`modules` ← `self.plan.target_method.critical_modules` {empty list if absent}

`code` ← READFILE("main.py")
for all `m` ∈ `modules` **do**
  if not ASTSEARCH(`m`, `code`) **then**
    return (FALSE, "Critical module missing: " + `m`)
  end if
end for
return (TRUE, "All listed modules present")
end procedure

Verdict. Algorithm 8 gives the exact rule. The L2 decision is the base verdict. It is capped at `MET_PARTIALLY` if `metrics.json` is missing, and set to `FAILED` if L3 fails (reproduction mode only). The L1 result (`q_pass`) is computed and logged but not used. In the notation of the main text, Pass $= \mathbb{1}[\text{L2} = \texttt{MET_ALL}] \wedge \mathbb{1}[\text{L3}]$, up to the `metrics.json` cap.

Algorithm 8 Auditor verdict

procedure EXECUTEEVALUATION*()*
`m_data` ← LOADMETRICS()
`q_pass`, _ ← RUNDYNAMICQUANTITATIVECHECK(`m_data`) {logged only; not used below}

`decision` ← RUNUNIVERSALQUALITATIVECHECK()
if `decision` == MET_ALL **and** `m_data` is NULL **then**
  `decision` ← MET_PARTIALLY
end if
if not `self.discovery` **then**
  `l3_pass`, _ ← RUNCODEALIGNMENTCHECK()
  if not `l3_pass` **then**
    `decision` ← FAILED
  end if
end if
return `decision`
end procedure

Regex failure tags (hints only). `experiment_taxonomy.py` scans auditor output and reports for the keyword patterns below and emits candidate M1–M5 tags. These tags are brittle keyword heuristics and **are not the labels reported in this paper**. They were shown to the annotator as hints only (Appendix D). The reported labels were set by the annotation agent from the rubric (Appendix D) and verified by the authors; they were not copied from these tags.

```
FAILURE_CATEGORIES = {
    "METHOD_COLLAPSE": {
        "signals": ["oracle", "upper bound", "never started", "substituted"]
    },
    "SILENT_DEGRADE": {
        "signals": ["from.scratch", "random init", "degenerate.*solution"]
    },
    "SCALE_INVERSION": {
        "signals": ["baseline.*outperform", "direction.*reversal"]
    },
    "KEY_MISMATCH": {
        "signals": ["metrics.json.*missing", "key.*not.*match"]
    },
    "INCOMPLETE": {
        "signals": ["not all condition", "time budget exhaust"]
    }
}
```

A.4 THE YAML REFERENCE CONTRACT

Provenance and known errors. Contracts are generated automatically by GPT-4.1 from the source paper (`generation_metadata.generator_model`). They were *not* verified by humans before the runs. The field `validation_status: complete` means only that the file passed schema validation in `gen_exp_plan.py` (required fields present, defaults auto-filled where applicable); it does not certify the plan's content against the paper. We therefore call them *reference contracts*, not ground truth. They can contain errors. For example, the first entry of `expected_paper_conclusions` in the DPR contract below says that in-batch negatives alone are "more sample-efficient than hard negatives". This misstates the original paper, which reports that adding a BM25 hard negative improves retrieval. We leave the contract exactly as it was used in the runs rather than silently correcting it. Contract quality is discussed as a threat to validity in the main text.

Listing 2: Reference contract used for the DPR run (`bm_001_dpr_dpr.yaml`), reproduced verbatim. Field consumers are listed in Table 7.

```
topic: Dense Passage Retrieval for Open-Domain Question Answering
paper_metadata:
  title: Dense Passage Retrieval for Open-Domain Question Answering
  authors: Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, ...
  year: 2020
  venue: EMNLP
  domain: nlp_retrieval
  task_type: open_domain_qa_retrieval
reproduction_goal: >
  Evaluate whether an AI agent can reproduce the main experimental
  conclusions of DPR: that dense dual-encoder retrieval outperforms
  BM25 on open-domain QA retrieval metrics, using reduced-scale
  reproduction on canonical datasets.
key_claims:
  - Dense dual-encoder retrieval (DPR) substantially outperforms BM25
    on open-domain QA retrieval metrics such as Recall@k and MRR.
  - Dense retrieval provides stronger downstream evidence quality for
    QA than sparse lexical retrieval alone.
datasets:
  - NaturalQuestions
  - TriviaQA
baselines:
  - BM25 (sparse lexical retrieval, e.g., via Pyserini or Elasticsearch)
target_method:
  name: Dense Passage Retriever (DPR)
  description: A dual-encoder dense retrieval model that encodes
    questions and passages into a shared embedding space for
    efficient top-k retrieval via maximum inner product search.
  codebase: facebookresearch/DPR
ablations:
  - DPR with different negative sampling strategies
  - DPR with reduced training data
metrics:
  - recall_at_20
  - recall_at_100
  - mrr
expected_metric_values:
  - metric_name: recall_at_20
    expected_value: 0.78
```

```yaml
    reasonable_range: [0.73, 0.8]
    higher_is_better: true
  - metric_name: recall_at_100
    expected_value: 0.85
    reasonable_range: [0.8, 0.88]
    higher_is_better: true
  - metric_name: mrr
    expected_value: 0.32
    reasonable_range: [0.29, 0.34]
    higher_is_better: true
expected_paper_conclusions:
  - The paper shows that a dense retriever trained with in-batch negatives
    alone reaches strong open-domain retrieval accuracy without curated
    hard-negative mining, which is more sample-efficient than hard negatives.
  - The paper further demonstrates that DPR transfers to out-of-domain
    datasets, preserving its retrieval advantage over sparse BM25 in
    unseen QA domains.
success_conditions:
  conclusion_correctness: true
  metric_direction_consistency: DPR must outperform BM25 on all
    primary retrieval metrics (recall_at_20, recall_at_100, mrr) in
    the same direction as the original paper.
  result_closeness_tolerance: 0.05
failure_conditions:
  - BM25 outperforms or matches DPR on primary retrieval metrics.
  - Metric differences are outside the reasonable range or opposite
    in direction to the paper.
  - Incorrect evaluation setup (e.g., wrong negatives, inconsistent
    preprocessing) invalidates comparison.
compute_budget:
  gpu_type: NVIDIA GeForce RTX 4070 Laptop GPU
  gpu_memory_gb: 8
  max_gpu_hours: 18.0
  cpu_cores: 8
  ram_gb: 32
time_budget:
  environment_setup_hours: 2.0
  repo_selection_and_code_reading_hours: 2.0
  dataset_parsing_and_section_segmentation_hours: 2.0
  baseline_pipeline_adaptation_hours: 2.0
  proposed_method_implementation_hours: 2.5
  debugging_and_memory_optimization_hours: 2.0
  baseline_training_and_evaluation_hours: 2.0
  proposed_method_training_and_evaluation_hours: 2.5
  ablation_runs_hours: 1.0
  result_aggregation_and_visualization_hours: 1.0
  contingency_buffer_hours: 1.0
risks:
  - Incorrect negative sampling or passage index construction may invalidate results.
  - Incompatible preprocessing between DPR and BM25 could bias metrics.
  - GPU memory limits may require smaller batch sizes or model variants, affecting metric
  ↪ values.
  - Dataset splits or answer string matching may differ from the original, impacting recall.
reproducibility_notes:
  Reduced-scale reproduction uses DPR-base model, smaller batch sizes, and fewer epochs to
  ↪ fit within compute budget. Pretrained checkpoints may be used for evaluation if full
  ↪ training is infeasible. Metric values are expected to be within 5% of the original
  ↪ papers results, but directionality and relative ranking between DPR and BM25 are the
  ↪ main criteria for success.
generation_metadata:
  generated_at: '2026-06-01T16:42:13.157186'
  generator_model: gpt-4.1
  benchmark_id: bm_001_dpr
  benchmark_name: DPR
  priority_tier: P1
  difficulty: 2
  reproduction_mode: reduced_scale_reproduction
  retry_count: 0
  validation_status: complete
```

Who reads which field. Table 7 lists, for each field, the component that actually reads it in the current code. Most fields reach the process only as text inside agent prompts. Only `datasets` and `target_method` (L1), `key_claims`, `expected_paper_conclusions` and `failure_conditions` (L2), and `critical_modules` (L3) are read by the auditor; the rows below were checked against the current implementation.

Table 7: Contract fields and the components that read them in the current implementation.

Field	Read by	How it is used
`topic`, `paper_metadata`, `reproduction_goal`	Agent (prompt)	Scope and reference-code search in the Intent/Research steps
`datasets`	Agent (prompt); Auditor L1	L1 checks that each dataset has an entry in `metrics.json`
`target_method`	Agent (prompt); Auditor L1	L1 checks that the target method has an entry under each dataset
`target_method.critical_modules` (optional)	Auditor L3	Names that must appear in the AST of `main.py`; if absent, L3 passes vacuously
`baselines`, `metrics`, `ablations`	Agent (prompt)	What to implement and report; not checked by the auditor
`key_claims`	Auditor L2	Claims the LLM judge evaluates against the report
`expected_metric_values`, `success_conditions`	Agent (prompt)	Used in the agent's own analysis step (Alg. 2); not checked numerically by the auditor
`expected_paper_conclusions`, `failure_conditions`	Auditor L2; Agent (prompt)	The L2 judge prompt carries them verbatim (Listing 1); they also reach the agent as part of the plan text
`compute_budget`, `time_budget`	Agent runtime	Mapped to per-step timeouts
`risks`, `reproducibility_notes`	Agent (plan file)	Contextual guidance in the plan; not read by any auditor check
`generation_metadata`	Bookkeeping	Provenance and versioning

A.5 File inventory

Table 8 lists the main modules.

Table 8: Main system files.

File Name	Lines	Role
`ralph_github.py`	∼1246	Agent workflow, improvement loop, ablation loop
`verify_reproduction.py`	∼519	Post-hoc auditor (L1, L2, verdict)
`alignment_verifier.py`	∼266	L3 module presence check
`experiment_taxonomy.py`	∼220	Regex failure tags (hints only)
`ablation_scoring.py`	∼280	Scoring of dimensions A–F
`experiment_integrator.py`	∼390	Execution history, metric tracking, composite score

B Scoring Protocol and Per-Dimension Results

Dimensions. Each workspace is scored on six dimensions (0–100). Table 9 lists them and states how each is scored. Dimensions A–D are computed from workspace artifacts and are the only ones that enter the composite score. Dimensions E and F are scored by an LLM and are reported separately as diagnostics.

Table 9: Scoring dimensions. $S_{\text{comp}} = 0.20\,s_A + 0.25\,s_B + 0.25\,s_C + 0.30\,s_D$. The weights were set by the authors before the runs and were not tuned; we have not analysed their sensitivity.

	Dimension	Scorer	In composite	Definition
A	Design	Rule-based	0.20	Repository structure and design over the run's code files
B	Reliability	Rule-based	0.25	Stages completed without unrecovered errors; run-time exceptions and crashes
C	Rigor	Rule-based	0.25	Rigor of the evaluation protocol (baselines set up, metrics defined)
D	Completeness	Rule-based	0.30	Fraction of contracted datasets × methods with reported metrics
E	Alignment	Rule-based	–	Module existence, blueprint coverage, and condition coverage against the contract (`score_alignment`)
F	LLM Review	LLM	–	Holistic LLM review of the report and pipeline; gated behind `EXPERIMENT_LLM_ANALYSIS`=1

Dimensions A–E are computed by rule-based checks over workspace artifacts with no LLM call; only F invokes an LLM, and only when the `EXPERIMENT_LLM_ANALYSIS` flag is set.

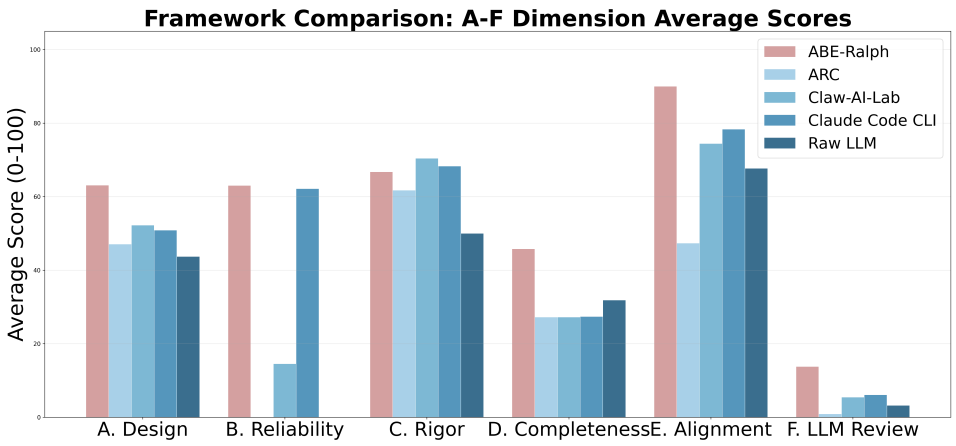

Figure 5: Average scores on dimensions A–F (0–100) per system, single run per task. A: Design, B: Reliability, C: Rigor, D: Completeness (rule-based, enter the composite); E: Alignment, F: LLM Review (LLM-scored, diagnostic only).

Main-table score vs. ablation score. Both tables use the same scoring formula SCORE-WORKSPACEMULTIDIM with the same weights over dimensions A–D; they differ only in the task set scored. The ablation ran on the 30 reproduction tasks (Full $= 44.2$), whereas RQ1 ran on the 21-task comparison set ($ABE - Ralph = 58.8$; Section 5.2 of the main text). **The two numbers are therefore not comparable**, and ablation deltas should be read only against the ablation's own Full configuration. Because the formula is the same, re-scoring one task set with the other's runs is not meaningful; no such re-scoring is performed here.

Per-dimension results. Figure 5 reports the average score on each dimension for each system. All systems are run once per task with different underlying models and budgets (see the main text), so the differences below are descriptive.

- **Reliability (B).** ABE-Ralph (63) and Claude Code CLI (62) score similarly. Raw LLM and ARC score near zero, mostly because their generated scripts fail on missing packages (`ModuleNotFoundError`) or wrong paths and they have no execution-feedback loop to repair them. The similarity between ABE-Ralph and Claude Code is expected, since ABE-Ralph runs on top of Claude.
- **Completeness (D).** Completeness is low for every system: ABE-Ralph reaches 46 and all others are below 32. This is consistent with the high rate of incomplete execution (M5) in our labels (50.0% of runs).
- **Alignment (E) and LLM Review (F).** ABE-Ralph scores higher on Alignment (90 vs. 78 for Claude Code CLI). Alignment is rule-based — it scores module existence, blueprint coverage and condition coverage against the contract — so this is a programmatic property, not a judge's opinion. It is the one dimension on which ABE-Ralph benefits structurally from the contract: ABE-Ralph sees the contract at every workflow step, while the baselines receive only the plan-derived configuration. Part of the gap may therefore reflect the workflow's contract-centric structure rather than better underlying methodology. All systems score below 15 on F, which indicates that reports are judged weak even when the code runs.

B.1 POSITIONING RELATIVE TO EXISTING AGENTIC SYSTEMS

Table 10 contrasts the *design goals* of ABE-Ralph with those of representative systems. It is a qualitative comparison and does not report measured performance.

Table 10: Qualitative comparison of design goals.

Dimension	AI Scientist / AutoResearchClaw	SWE-agent / OpenHands	Human Repro. Challenge	ABE-Ralph (Ours)
Primary goal	Open-ended discovery	Code repair and debugging	Manual replication	Contract-guided reproduction with post-hoc audit
Success check	Generated paper and metrics	Unit tests	Human review	Coverage check, LLM judge, module presence
Constraint source	Free-form idea	Tests	Checklist	LLM-generated YAML contract
Shortcut handling	Not an explicit design target	Outside test scope	Human expert	Flagged post hoc; not prevented (56.7% of runs still labelled)
Failure reporting	Free-text logs	Stack traces	Text report	Auditor verdict + manual M1–M5 labels

B.2 The staged workflow

Table 11 describes each stage. Timeouts are derived from the contract's time budget.

Table 11: Stages of the ABE-Ralph workflow.

Step	Name	Description
1	Intent Discovery	Parse the contract and extract goals, constraints and metrics.
1.5	Dataset Verification	Search for the specified real datasets. Synthetic data is discouraged in the prompt; if no real dataset is confirmed the step emits a warning and the run continues.
2	Repo Search & Selection	Locate reference implementations from official or well-known sources.
3	Architecture Blueprint	Write `blueprint.md` describing the model and the interfaces in `main.py`.
4a	Smoke Test	Run on 5–10 samples to check data paths, memory and CUDA setup.
4b	Main Execution	Run the full pipeline and write `metrics.json`. OOM handling is left to the agent under prompt guidance.
5	Analysis & Reporting	Write `experiment_result.md` comparing results with the contract.
5.5	Output Fallback	No fallback step exists in the current implementation; a missing `metrics.json` is left to the report step and later to the auditor's L1 check.
6	Skill Extraction	Save reusable code utilities from the run.

C Discovery Mode

ABE-Ralph also has a discovery mode for tasks without a source paper (Table 5). The contract then specifies a problem statement, allowed resources and an external evaluator (`evaluator.py`) instead of expected results. The agent reads the problem description, searches the literature rather than reference code, and optimizes the evaluator score; L3 is skipped (Algorithm 8). Discovery mode is not part of the main study. We report only a preliminary trial in Appendix G.

D Labelling Protocol and Taxonomy

Runs. All 30 runs analysed in this paper are ABE-Ralph runs in reproduction mode, one run per paper. Table 13 lists the papers by domain.

Annotation. Labels were assigned by an LLM-based annotation agent, after reading, for each run, the final report, `metrics.json`, `main.py` and the execution logs, and were then verified by the authors. The annotation agent applied the rubric below and saw the regex tags from `experiment_taxonomy.py` only as hints. Labels were assigned in a single pass and we have not measured inter-annotator agreement. All labels are kept in `labels.csv`.

Unit of labelling. A label is attached to an *observed fact* in a run, for example "the DDPM baseline was never executed". A run may receive several labels if several distinct facts apply. **One fact receives exactly one label.** If a fact fits several categories, the first matching rule below wins.

Boundary rules.

1. **Genuine reproduction failure is not a hallucination.** If the method and protocol were implemented as specified and the result simply does not match the paper, and the report says so, no label is assigned. Such runs are noted separately (Improved DDPM is the only case).
2. **Disclosed deviations are not hallucinations.** A deviation from the contract, such as a smaller model or a missing dataset, is labelled only if the report does not clearly disclose it or presents the affected claim as reproduced.
3. **Auditor-interface failures are labelled M4.** They do not affect the scientific content of the run, and we report them separately because they cause false alarms in automatic checks.

Table 12: Taxonomy of methodological hallucinations with decision rules. Counts are over 30 runs; labels overlap.

Category	Decision rule (label the fact if...)	Representative case	Runs
M1 Method Integrity Collapse	...the core method, or the core comparison on which the paper's central claim rests, is absent or replaced by a trivial substitute, yet results are presented for it.	**RAG:** retrieval failed and the generator answered from parametric memory, while the numbers were reported as RAG results.	3
M2 Silent Protocol Degradation	...the training, evaluation protocol or success criterion was changed from the contract without disclosure.	**PEGASUS:** trained from scratch instead of fine-tuning the pre-trained checkpoint.	8
M3 Scale-Driven Conclusion Inversion	...at the reduced scale the paper's ranking or conclusion is inverted, and the report does not attribute the inversion to scale or still claims reproduction.	**ViT:** CNN baselines outperform ViT without large-scale pre-training.	3
M4 Quantitative Key Mismatch	...results exist but are written under file or key names the auditor cannot match (auditor-interface failure).	**GraphSAGE:** `acc` vs. `accuracy`.	2
M5 Incomplete Execution	...required conditions, datasets or baselines (other than the core comparison, which is M1) were not run, and this is not clearly disclosed.	**DDIM:** 4 of 12 planned conditions completed, single seed.	15

Relation to the hardware setting. All runs use a single 8 GB laptop GPU. Several papers, such as ViT, MoCo and PPO on MuJoCo, cannot be reproduced at full scale under this budget, which raises the base rate of M3 and M5. We treat these labels as describing how the agent *behaves* under realistic resource limits, in particular whether it discloses the resulting deviations, and not as intrinsic properties of the papers.

Note on domains: the 12 domain labels in Table 13 are a coarse grouping used for presentation; the contracts record a finer per-task `paper_metadata.domain` value (about 18 distinct values). The grouping does not affect any statistic in this paper.

E Per-Category Breakdown

Of the 30 ABE-Ralph runs, 17 (56.7%) received at least one label and 13 (43.3%) received none. The runs carry 31 labels in total, and 11 runs carry more than one. The lists below are generated from `labels.csv`.

- **M1 (Method Integrity Collapse)**, 3 runs: RAG, SimCLR, DDIM.

Table 13: The 30 papers, grouped by domain.

Domain	Papers
Open-domain retrieval	DPR, RAG, ColBERT
Language-model pre-training / fine-tuning	BERT, RoBERTa, LoRA, Prefix-Tuning
Summarization and long documents	PEGASUS, LED, Longformer, TextRank
Image classification	ResNet, ViT
Representation learning	CLIP, SimCLR, MoCo
Segmentation	U-Net
Object detection	Faster R-CNN, YOLO
Anomaly detection	PatchCore, PaDiM
Diffusion models	DDPM, DDIM, Improved DDPM
Reinforcement learning	DQN, PPO
Graph learning	GCN, GraphSAGE
Time-series forecasting	Informer, Autoformer

- **M2 (Silent Protocol Degradation)**, 8 runs: ColBERT, BERT, PEGASUS, ViT, SimCLR, Faster R-CNN, PPO, GraphSAGE.
- **M3 (Scale-Driven Conclusion Inversion)**, 3 runs: PEGASUS, ViT, U-Net.
- **M4 (Quantitative Key Mismatch)**, 2 runs: RoBERTa, GraphSAGE.
- **M5 (Incomplete Execution)**, 15 runs: RAG, ColBERT, BERT, RoBERTa, PEGASUS, LED, ViT, MoCo, U-Net, YOLO, PatchCore, DDIM, Improved DDPM, PPO, GraphSAGE.
- **No hallucination label**, 13 runs: DPR, LoRA, Prefix-Tuning, Longformer, TextRank, ResNet, CLIP, PaDiM, DDPM, DQN, GCN, Informer, Autoformer.
- **Genuine reproduction failure noted (not counted as hallucination)**: Improved DDPM.

Aggregate statistics ($N = 30$, labels overlap).

- No label: 13 (43.3%)
- M1 Method Integrity Collapse: 3 (10.0%)
- M2 Silent Protocol Degradation: 8 (26.7%)
- M3 Scale-Driven Conclusion Inversion: 3 (10.0%)
- M4 Quantitative Key Mismatch: 2 (6.7%)
- M5 Incomplete Execution: 15 (50.0%)

Sensitivity to the disclosure rule. 5 runs carry only an M5 label. Under boundary rule 2, each of these would become unlabelled if its report had clearly disclosed the missing parts. The headline rate of 56.7% is therefore sensitive to how strictly "clear disclosure" is judged. Without access to the run logs in this revision we do not re-audit whether these five reports (LED, MoCo, YOLO, PatchCore, Improved DDPM) disclosed the missing parts; the main text already expresses the resulting uncertainty as a range (40.0%%–56.7%%). The labels are kept exactly as recorded.

F Per-Run Case Studies

The status line of each case study must match `labels.csv`; `make_taxonomy_assets.py` checks this automatically.

1. DPR (Dense Passage Retrieval) *Status: CLEAN.* Metrics for NaturalQuestions and TriviaQA are present, and the reproduced numbers support the conclusion that DPR outperforms BM25. The known error in this run's contract (Appendix A.4) concerns `expected_paper_conclusions`, which is not a claim the run was asked to establish.

2. RAG (Retrieval-Augmented Generation) *Status: HALLUCINATION (M1 + M5).*

- **M1:** Retrieval failed and the generator fell back to parametric knowledge (EM 1.8% vs. 42.1% with successful retrieval). The numbers were nevertheless reported as RAG performance, so the retrieval component's contribution was never isolated.
- **M5:** Metrics for NaturalQuestions and WebQuestions are missing because index construction failed.

3. ColBERT (Contextualized Late Interaction over BERT) *Status: HALLUCINATION (M2 + M5).*

- **M2:** Hard-negative mining was omitted from training without disclosure. ColBERT depends on hard negatives, and the model failed to discriminate relevant from irrelevant passages.
- **M5:** The TREC Deep Learning benchmark was not run.

4. BERT (Pre-trained Language Model) *Status: HALLUCINATION (M2 + M5).*

- **M2:** Fine-tuning was done on SST-2 only, a task chosen by the agent, instead of the contract's MNLI, QQP and MRPC. SST-2 is itself a GLUE task, so the substitution changes the evaluation protocol but does not remove the method. (Previously labelled M1; relabelled under the rubric.)
- **M5:** The remaining GLUE tasks were not run.

5. RoBERTa (Robustly Optimized BERT) *Status: HALLUCINATION (M4 + M5).*

- **M4:** The aggregation script wrote `metrics.json` under a non-standard filename, so L1 could not find it.
- **M5:** The full GLUE suite was not evaluated.

6. LoRA (Low-Rank Adaptation) *Status: CLEAN.* All required GLUE and instruction-tuning metrics are present. The conclusion that LoRA matches full fine-tuning with far fewer trainable parameters is supported.

7. Prefix-Tuning (Continuous Prompt Tuning) *Status: CLEAN.* Continuous prefix parameters were set up and compared against full fine-tuning, with stable convergence.

8. PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) *Status: HALLUCINATION (M2 + M3 + M5).*

- **M2:** The model was trained from scratch for 3 epochs on 50K samples instead of fine-tuning the pre-trained gap-sentence (GSG) checkpoint.
- **M3:** The ranking inverted: PEGASUS scored -32% vs. BERT and -55% vs. BART, contrary to the paper; the run record gives no indication that the report attributed the inversion to the reduced scale, so we keep M3.
- **M5:** The XSum benchmark was not run.

9. LED (Longformer-Encoder-Decoder) *Status: HALLUCINATION (M5).* Target-method metrics on the arXiv long-document dataset are missing, so the long-context summarization claim was not tested.

10. Longformer (Long-Document Transformer) *Status: CLEAN.* Sliding-window plus global attention masks were implemented correctly, and the expected efficiency–performance trade-off was recovered.

11. TextRank (Graph-based Unsupervised Summarization) *Status: CLEAN.* The graph-based extractive summarizer was implemented and ROUGE was computed correctly.

12. ResNet (Deep Residual Learning) *Status: CLEAN.* The advantage of residual over plain CNNs was reproduced on CIFAR-10/100 with proper residual connections and learning-rate schedules.

13. ViT (Vision Transformer) *Status: HALLUCINATION (M2 + M3 + M5).*

- **M2:** ViT was trained from scratch on a small dataset without the large-scale pre-training the paper assumes, costing about 20 percentage points of accuracy. The deviation was not disclosed.

- **M3:** At this scale CNN baselines outperformed ViT, the opposite of the paper's conclusion.
- **M5:** The ImageNet subset was not evaluated.

14. CLIP (Contrastive Language-Image Pre-training) *Status: CLEAN*. Zero-shot classification was evaluated with appropriate prompt templates, and the expected zero-shot transfer behaviour was recovered.

15. SimCLR (Simple Framework for Contrastive Learning) *Status: HALLUCINATION (M1 + M2).*

- **M1:** The baseline encoders were queued but never run, so the core comparison (SimCLR vs. prior unsupervised methods) is absent.
- **M2:** The report replaced the success criterion with "above chance" performance instead of superiority over baselines. (Previously labelled M3; this is a criterion change, not a scale effect. The former M5 label described the same fact as M1 and was merged.)

16. MoCo (Momentum Contrast) *Status: HALLUCINATION (M5).* The ImageNet-subset linear-probe evaluation is missing, so the primary claim was not tested.

17. U-Net (Convolutional Networks for Biomedical Image Segmentation) *Status: HALLUCINATION (M3 + M5).*

- **M3:** With limited data, the reduced-depth variant outperformed the full U-Net, contradicting the paper's claim about encoder–decoder depth; the run record gives no indication that the report flagged this as a scale effect, so we keep M3.
- **M5:** The ISBI segmentation benchmark was not run.

18. Faster R-CNN (Towards Real-Time Object Detection) *Status: HALLUCINATION (M2).* The deviation is an evaluation-protocol one: mAP was evaluated on a detector trained for only 8% of the training schedule (the run record confirms RPN convergence but not model convergence), i.e. a non-standard mAP evaluation protocol, and the deviation was not reported. The baseline comparison and the ablation run were also absent.

19. YOLO (Real-Time Object Detection) *Status: HALLUCINATION (M5).* Target-method metrics on PASCAL VOC are missing, so the speed–accuracy trade-off is incomplete.

20. PatchCore (Industrial Anomaly Detection) *Status: HALLUCINATION (M5).* Pixel-level localization metrics (AUROC/AUPRO) are missing.

21. PaDiM (Patch Distribution Modeling) *Status: CLEAN*. Mahalanobis-distance modelling of patch embeddings was implemented and verified on MVTec AD.

22. DDPM (Denoising Diffusion Probabilistic Models) *Status: CLEAN*. The diffusion pipeline was reproduced with the correct beta schedule and sampling loop, and FID was reported.

23. DDIM (Denoising Diffusion Implicit Models) *Status: HALLUCINATION (M1 + M5).*

- **M1:** The DDPM sampling baseline was never run, so the central speed-up claim over DDPM was not tested.
- **M5:** Only 4 of 12 planned conditions were completed, with a single seed.

24. Improved DDPM (Improved Denoising Diffusion) *Status: HALLUCINATION (M5).* Metrics for the ImageNet subset are missing. *Genuine reproduction failure (not labelled):* with learned variance, training was unstable and the expected improvement over DDPM did not appear. Because the method was implemented as specified and the failure is recorded in the run report, this falls under boundary rule 1 (a genuine failure, not a hallucination); an M1 label would mischaracterize it. (Previously labelled M1.)

25. DQN (Deep Q-Networks) *Status: CLEAN*. The replay buffer and target-network updates were implemented correctly, and the expected performance was recovered.

26. PPO (Proximal Policy Optimization) *Status: HALLUCINATION (M2 + M5).*

- **M2:** All continuous-control runs (MuJoCo HalfCheetah, Hopper, Walker2d) crashed, 0 of 9 completed. The report nevertheless claimed continuous-control success on the basis of the discrete CartPole-v1 environment.
- **M5:** No baseline results (Vanilla PG, TRPO) were produced.

27. GCN (Graph Convolutional Networks) *Status: CLEAN*. Graph convolution, adjacency normalization and the standard splits were set up correctly, and node-classification accuracy on Cora and Citeseer was recovered.

28. GraphSAGE (Inductive Representation Learning on Large Graphs) *Status: HALLUCINATION (M2 + M4 + M5).*

- **M2:** The inductive setting was misconfigured as transductive, which removes the basis of the inductive-generalization claim.
- **M4:** Inconsistent metric keys (`acc` vs. `accuracy`) prevented L1 from matching the results.
- **M5:** The Reddit and PPI datasets were not run.

29. Informer (Long Sequence Time-Series Forecasting) *Status: CLEAN*. ProbSparse attention and the generative long-horizon decoder were implemented, and the efficiency gains were recovered.

30. Autoformer (Decomposition Transformers) *Status: CLEAN*. Series decomposition and Auto-Correlation were reproduced and improved over a vanilla Transformer.

F.1 SUMMARY TABLE

Table 14: Per-run labels for the 30 ABE-Ralph reproduction runs (generated from `labels.csv`). A run can carry several labels; one observed fact receives one label (Appendix D).

#	Method	M1	M2	M3	M4	M5	Observed issue
1	DPR	–	–	–	–	–	All required metrics present for NaturalQuestions and TriviaQA; DPR ¿ BM25 supported by the reproduced numbers.
2	RAG	✓	–	–	–	✓	**M1:** retrieval failed and the generator fell back to parametric memory while results were reported as RAG. **M5:** NaturalQuestions/WebQuestions metrics missing.
3	ColBERT	–	✓	–	–	✓	**M2:** hard-negative mining silently omitted. **M5:** TREC Deep Learning metrics missing.
4	BERT	–	✓	–	–	✓	**M2:** evaluated on SST-2 only instead of the specified MNLI/QQP/MRPC. **M5:** remaining GLUE tasks missing.
5	RoBERTa	–	–	–	✓	✓	**M4:** metrics written under a non-standard filename. **M5:** full GLUE suite missing.
6	LoRA	–	–	–	–	–	All required GLUE and instruction-tuning metrics present; claim supported.
7	Prefix-Tuning	–	–	–	–	–	Prefix parameters set up correctly and compared with full fine-tuning.
8	PEGASUS	–	✓	✓	–	✓	**M2:** trained from scratch (3 epochs; 50K samples) instead of fine-tuning the GSG checkpoint. **M3:** ranking inverted (-32% vs. BERT). **M5:** XSum missing.
9	LED	–	–	–	–	✓	**M5:** target-method metrics missing on arXiv.

#	Method	M1	M2	M3	M4	M5	Observed issue
10	Longformer	–	–	–	–	–	Sliding-window + global attention implemented; expected trade-off recovered.
11	TextRank	–	–	–	–	–	Graph-based extractive summarization implemented; ROUGE computed correctly.
12	ResNet	–	–	–	–	–	Residual vs. plain CNN advantage reproduced on CIFAR-10/100.
13	ViT	–	✓	✓	–	✓	**M2:** trained from scratch without large-scale pre-training (≈20-pt drop). **M3:** CNN outperforms ViT at reduced scale. **M5:** ImageNet subset missing.
14	CLIP	–	–	–	–	–	Zero-shot classification evaluated with proper prompt templates.
15	SimCLR	✓	✓	–	–	–	**M1:** baseline encoders queued but never run; the core comparison is absent. **M2:** success criterion silently weakened to "above chance".
16	MoCo	–	–	–	–	✓	**M5:** ImageNet-subset linear-probe metrics missing.
17	U-Net	–	–	✓	–	✓	**M3:** reduced-depth variant beats full U-Net under limited data. **M5:** ISBI benchmark missing.
18	Faster R-CNN	–	✓	–	–	–	**M2:** mAP is evaluated on a detector trained only ∼8% of the schedule (far from convergence); per the original records this is the non-standard mAP evaluation protocol that caused the silent divergence (the baseline comparison and the ablation are also absent).
19	YOLO	–	–	–	–	✓	**M5:** target-method metrics missing on PASCAL VOC.
20	PatchCore	–	–	–	–	✓	**M5:** pixel-level AUROC / AUPRO missing.
21	PaDiM	–	–	–	–	–	Mahalanobis modelling of patch embeddings verified on MVTec AD.
22	DDPM	–	–	–	–	–	Correct beta schedule; sampling loop; FID reported.
23	DDIM	✓	–	–	–	✓	**M1:** DDPM baseline never run; speed-up claim untested. **M5:** 4 of 12 planned conditions completed; single seed.
24	Improved DDPM	–	–	–	–	✓	**M5:** ImageNet-subset metrics missing. *Note:* the lack of a quality gap over DDPM is a genuine reproduction failure and is not labelled.
25	DQN	–	–	–	–	–	Replay buffer and target network correct; expected CartPole/Atari performance.
26	PPO	–	✓	–	–	✓	**M2:** all 9 MuJoCo runs crashed; report claimed continuous-control success from CartPole-v1. **M5:** no baseline (Vanilla PG; TRPO) data.
27	GCN	–	–	–	–	–	Normalized adjacency and standard splits; accuracy recovered on Cora/Citeseer.
28	GraphSAGE	–	✓	–	✓	✓	**M2:** inductive setting misconfigured as transductive. **M4:** `acc` vs. `accuracy` key mismatch. **M5:** Reddit/PPI missing.
29	Informer	–	–	–	–	–	ProbSparse attention and generative decoder implemented; efficiency gains recovered.
30	Autoformer	–	–	–	–	–	Series decomposition and Auto-Correlation reproduced; gains over vanilla Transformer.

G PRELIMINARY DISCOVERY-MODE TRIAL (NATUREBENCH)

We ran ABE-Ralph in discovery mode, together with the four RQ1 baselines, on 16 tasks from NatureBench (Wang et al., 2026) spanning biology, chemistry, physics, graph/CS and causal ML domains. This trial is separate from the main study: each system ran once per covered task, coverage differed across systems, and we do not compare against any SOTA reference. Table 15 reports, per system, the number of tasks for which the run produced at least one evaluator-required indicator

and the total number of indicators produced. Eight of the 16 tasks produced no indicators from any system; the dominant bottleneck cited in the run records is the 1-hour per-task timeout. We draw no conclusions from this trial in the main text.

Table 15: Preliminary discovery-mode trial on 16 NatureBench tasks (single run per covered task; counts from `NATUREBENCH_COMPARISON.md`).

System	Tasks with $\geq$1 indicator	Total indicators
Claude Code CLI	6	97
ABE-Ralph	3	36
Claw-AI-Lab	1	4
ARC	0	0
Raw LLM	0	0

H Discussion

Software viability and scientific validity diverge. The per-dimension analysis (Appendix B) shows that general-purpose coding agents score well on design and reliability. Our annotations show that even the highest-scoring system still hallucinates in most runs, so these scores do not translate into faithful experiments. Faced with roadblocks, agents optimize for a clean exit, not for a valid test of the claim.

Resource adaptation is the central difficulty. The prevalence of M5 and M2 indicates that the hard part of constrained reproduction is not writing code but deciding what to do at the compute ceiling. A useful agent should declare deviations, scale experiments so that the claim remains testable, or report that the claim cannot be tested under the budget. Agents that silently shrink the experiment do none of these.

Auditing should close the loop. In ABE-Ralph the agent decides for itself when its work is done, and the auditor only reports afterwards. Our results suggest that self-assessment is where fidelity is lost. Feeding audit verdicts back to the agent, or letting them block a run from being reported as complete, is the most direct next step.

Reporting remains weak. All systems score low on the LLM-judged report-quality dimension (Dimension F, Appendix B), which suggests that agents struggle to turn their runs into coherent, contextualized reports. A roadmap is given in Appendix J.

Discovery mode. We also ran a preliminary trial of a discovery-mode variant on 16 NatureBench tasks (Wang et al., 2026). Because this trial has one run per task and partial per-system coverage, we report it only descriptively in Appendix C and draw no conclusions from it.

I Threats to Validity and Limitations

- **Confounded comparison.** The systems differ in task coverage (10–18 of 21 tasks), iteration budget, and decoding and tool-loop configuration, and their compute and token usage was not matched. In this environment all backbones resolved to the same commercial model family (`deepseek-v4-flash`), which removes the model-identity confound but not the others. The RQ1 gaps therefore cannot be attributed to the contract workflow alone.
- **Single runs.** Each system–task pair was run once, so we report no variance across seeds and no significance tests.
- **Post-hoc, coarse auditor.** The auditor does not intervene in the run. Its checks are coarse by design: L1 tests only whether results are present and does not contribute to the verdict, L2 is an uncalibrated LLM judge that shares the agent's backbone, and L3 matches module names and passed trivially in all runs because no 30 contract defines `critical_modules`. We do not measure detection accuracy.

- **Annotation.** Labels come from a single annotation pass by an LLM-based annotation agent, verified by the authors, and only ABE-Ralph runs were annotated, so we cannot compare hallucination rates across systems. The class boundaries (Section 4) involve judgment; in particular, the headline rate ranges from 40.0%% to 56.7%% depending on how "declared" is read for M5.
- **Reference contracts.** Contracts were generated by an LLM and not systematically verified, and at least one contains a factual error about its paper's findings (Section 3.2). Since auditing can be no better than its reference, human-verified contracts are necessary for high-stakes use.
- **Scores.** The composite score's four components and their weights are our own design. The alignment dimension (E, rule-based) and the LLM-judged dimension (F) are excluded from the composite but reported as diagnostics. The ablation scored a different task set than RQ1 (the formula is the same). None of these scores measure fidelity.
- **Scope of tasks and budget.** The tasks cover only machine learning, and all of them run on a single 8 GB GPU. The class frequencies, especially for M3 and M5, depend on this setting and may not transfer to other domains or budgets.

J Future Work

The preceding appendix sections (in particular the discussion in Appendix H) interpret the findings. Here we only outline directions suggested by the failure modes.

Moving checks into the loop. In the current prototype the auditor runs after the fact and its L1 result is not used. The most direct extension is to feed auditor findings back to the agent before it declares convergence, and to make L1 an actual numeric check against `expected_metric_values` with a tolerance. Both are simple to implement, but their effect has to be measured, including how often they cause false rejections such as the M4 cases.

Human-verified contracts. Because contracts are LLM-generated and can be wrong (Appendix A.4), a small human-verified contract set is a prerequisite for using this protocol in high-stakes auditing.

Resource-aware multi-agent designs. M5 is the most frequent label (50.0%), and many M2/M3 cases arise when the agent silently downscales under the 8 GB budget. A natural direction is a cooperative design Yuan et al. (2025) with separate roles:

- a **resource orchestrator** that monitors memory and time and applies pre-declared, disclosed scaling rules (e.g., gradient accumulation) instead of ad hoc changes;
- a **developer** agent that writes and runs code;
- an independent **auditor** agent, ideally from a different model family than the developer, that checks code and call graphs against the contract;
- a **red-teaming** agent that searches for data subsets or settings that would invalidate a reported result.

Whether such designs actually reduce the labelled failure modes is an empirical question we leave open.