SCIENTISTONE: VERIFIABLE AUTONOMOUS RESEARCH VIA CHAIN-OF-EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous research agents now produce competitive solutions and polished manuscripts, but their outputs can carry verifiability failures that presentation-focused evaluation misses: fabricated citations, unreproducible scores, and method descriptions that diverge from the code. These failures share a root: today's autonomous research systems are not built to trace their claims back to evidence. We address this with three contributions. First, *Chain-of-Evidence* (CoE), a verifiability framework under which every claim traces to its evidence source. Second, **ScientistOne**, an end-to-end autonomous research system that maintains evidence chains by construction across literature review, solution discovery, and paper writing. Third, CoE Integrity Audit, a post-hoc audit whose four integrity checks (score verification, specification violation, reference verification, and method–code alignment) apply the same criteria across systems. Across 75 papers from five systems on five frontier systems-research tasks, every baseline shows at least one systematic failure mode. Among the three baselines in the comparison, up to 24% of references are hallucinated, score verification passes in as few as 42% of papers, and method–code alignment ranges from 20% to 87%. **ScientistOne** is best or tied for best on all four checks, with none of its 340 distinct references hallucinated, 12/12 scores verified, and 14/15 method sections that match the code, while matching or exceeding the human expert solution on all five tasks. Against blinded human raters, the audit is right on all 48 score extractions and 219 of 220 passed references, and its method–code verdicts agree with the raters about as often as the raters agree with each other (92%). Our findings show that a deliberately simple discovery loop already reaches expert-level solutions, and we argue that the bottleneck in autonomous research is verifiability: CoE enables a paper to be audited before it is trusted, by both humans and machines.

1 INTRODUCTION

Large language models now run entire research workflows as autonomous agents, from literature review and hypothesis generation through experiments to a finished manuscript (Lu et al., 2024; Yamada et al., 2025; Weng et al., 2025; Tang et al., 2025; Schmidgall et al., 2025). On systems-optimization tasks their solutions match human experts (Cheng et al., 2025b; Novikov et al., 2025), a plain coding agent with no search scaffold reaches published state-of-the-art results on algorithm design problems (Liu et al., 2026e), and end-to-end pipelines have produced papers accepted at peer-reviewed workshops (Yamada et al., 2025).

An autonomous research system is a pipeline in which each stage consumes the output of the previous one: the hypothesis is based on the literature summary, the experiments are designed from the hypothesis, and the manuscript is written from the experimental results. An error introduced at any stage is preserved and often amplified. The resulting paper reads as internally coherent, because the same error recurs across sections, but that does not mean it is factually correct. Nevertheless, the evaluations commonly adopted in the community, automated review scores and benchmark leaderboards, measure how a paper reads and whether the scores are competitive, and neither checks whether individual claims trace to supporting evidence.

To quantify this gap, we audited 75 papers from five autonomous research systems on five benchmark tasks (Figure 1), and found evidence-chain failures in every baseline system: references that

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.

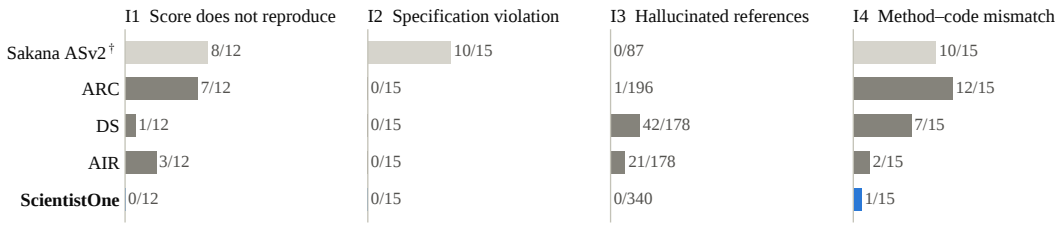

Figure 1: CoE Integrity Audit results on 75 papers from five autonomous research systems on five ADRS tasks. Share of papers whose reported score does not reproduce (I1), whose solution breaks the task rules (I2) or whose method section does not match the submitted code (I4), and share of distinct references that do not exist (I3). Counts for I1 to I3 of the four compared systems are confirmed by human review of every flagged case. I4 agrees with blinded raters about as often as they agree with each other (Table 2). †Sakana is shown but not compared with the others: its search asks the agent to design its own evaluation, which conflicts with the ADRS requirement that the task's evaluator be used (§6.1).

correspond to no real publication, up to 24% of a system's distinct references, method sections describing algorithms absent from the submitted code, scores that do not reproduce, and solution code that exploits the evaluator instead of solving the task. A concurrent audit of 117 agent-written papers finds the same failure modes, fabricated results, plan and execution mismatch, and fabricated references, once reviewers inspect the artifacts (Zhang et al., 2026). These failures share a cause: today's autonomous research systems are not designed to trace claims back to evidence.

We address this with *Chain-of-Evidence* (CoE), a verifiability framework for AI-driven research that defines what "verifiable" means for a research claim: **every claim must trace, through a recorded evidence chain, to a grounding source**, just as ACID defines what "reliable" means for a database transaction (Härder & Reuter, 1983). We make three contributions:

1. **Chain-of-Evidence** (§3): the principle, a claim taxonomy (citation, numerical, methodological, conclusion), and the evidence each type must trace to.
2. **`ScientistOne`** (§4): an end-to-end autonomous research system whose pipeline is built to satisfy CoE, including a Problem Investigator, a Discovery Engine, and a Paper Writer with a Claim Verifier.
3. **CoE Integrity Audit** (§5): a post-hoc audit of an AI-written research paper with four checks: Score Verification, Specification Violation, Reference Verification, and Method–Code Alignment.

We apply the audit to 75 papers from five systems on five ADRS tasks (Cheng et al., 2025b; Liu et al., 2026d) (§6). Every baseline fails at least one check systematically. **`ScientistOne`** is best or tied for best on all four checks and matches or exceeds the human expert solution on all five tasks with a deliberately simple discovery loop, and the audit's method–code verdicts agree with blinded human raters about as often as the raters agree with each other (§6.2). The bottleneck in autonomous research is therefore verifiability, and CoE is built to establish it.

2 Related Work

Autonomous research agents. The AI Scientist (Lu et al., 2024) pioneered end-to-end automation, Agent Laboratory (Schmidgall et al., 2025) and EvoScientist (Lyu et al., 2026) extend the end-to-end pipeline, and AlphaEvolve (Novikov et al., 2025) advances the solution-discovery stage with evolutionary code search. Curie (Kon et al., 2025a) validates experimental execution through reproducibility checks analogous to our I1 Score Verification, though it does not audit whether written claims faithfully reflect the validated results. We evaluate AI Scientist-v2 (Yamada et al., 2025) alongside three further systems, AutoResearchClaw (Liu et al., 2026b), DeepScientist (Weng et al., 2025), and AI-Researcher (Tang et al., 2025), which fail in different ways (§6.1). Across these designs, the ability to generate and run experiments has grown faster than the ability to check them.

Evaluation and benchmarks for autonomous research. The ADRS benchmark (Cheng et al., 2025b) collects frontier computer system research questions and is our primary evaluation testbed. EvoX (Liu et al., 2026c) and AdaEvolve (Cemri et al., 2026) achieve strong results on ADRS without

literature grounding or paper writing. Beyond ADRS, benchmarks for research agents (Chen et al., 2025b; Kon et al., 2025b; Chan et al., 2024; Chen et al., 2025a) test discovery, experimentation, ML engineering and open-ended research. MLR-Bench (Chen et al., 2025a) also checks whether the reported results are backed by the run, and finds fabricated or invalidated results in about 80% of cases. The others do not check the claims of a finished paper against the run that produced it.

Scientific integrity and provenance. Concurrent studies reach the same diagnosis. Zhang et al. (2026) run three off-the-shelf coding agents on 13 seeds and 117 papers, and find that manuscript-only review rewards plausible framing while artifact-aware review turns up fabricated results, plan and execution mismatch, and fabricated references, with results that do not match the artifacts in 5% to 77% of papers depending on the agent. Zhao et al. (2026) ask when an agent failure can be caught at all and show that the answer depends on which record the auditor is given: the declared configuration, a live trace, or the finished trace. Wang et al. (2026) audit 21 systems and name the pattern validation collapse, in which internal self-evaluation replaces independent validation. Rasheed et al. (2026) propose claim-level auditability as the design target, with provenance coverage and soundness among its measures. Agent-Native Research Artifacts (Liu et al., 2026a) replace the paper with a four-layer package (scientific logic, executable code, an exploration graph of failures, and evidence grounding every claim). The chain is built into the format, and the evaluation measures agent question answering and rubric-graded reproduction with an LLM as judge. For the method-code check, SciCoQA (Baumgärtner & Gurevych, 2026) collects 635 paper-code discrepancies, and the best of 22 LLMs detects 46.7% of the real ones. CoE shares the claim-level view of these studies and uses it on both sides: in a system built to satisfy it and in an audit that applies the same checks to five systems' outputs.

3 Chain-of-Evidence: A Framework for Research Verifiability

> ***Principle:*** *Every research claim must trace, through a recorded evidence chain, to a grounding source, and the evidence at each link must faithfully support the claim.*

A credible research claim must be backed by verifiable evidence. Completeness and correctness are separate requirements: a chain that ends in a real source can still misreport what the source says. ACID framework (Härder & Reuter, 1983) prescribes the properties a database must have and leaves the construction to the implementer. CoE plays the same role for research artifacts.

We define four primary claim types, each with its own evidence-chain requirement. The taxonomy covers the claim types that current tools can check. Qualitative observations and theoretical properties need domain expertise or subjective judgment that is harder to automate, and we leave them out. ***Citation claims*** ("Smith et al. showed X") require that the cited paper exists in a scholarly database and that its content is consistent with how it is described in the paper. ***Numerical claims*** ("achieves 87.3% on Prism") must trace from the reported value to a recorded output (an execution log, an experimental measurement, or a simulation result). ***Methodological claims*** ("we use a 3-layer MLP") must map from the method description to the code that implements it. These three types end directly in a source: a scholarly record, a recorded output, or an implementation. ***Conclusion claims*** ("our method is more sample-efficient than the baseline") are derived from other claims, so verifying one requires reasoning over the numerical and methodological claims that support it. A conclusion claim can contain a number. What defines it is that it is inferred from other claims, and the inference itself must hold. CoE says nothing about architecture: it defines the properties a verifiable artifact should have and leaves construction to the system. It also says nothing about authorship, since the same evidence chains are required of a human-written paper. The audit covers numerical claims, methodological claims and the existence of references. Citation support and conclusion claims are not covered (§7). In sciences where the evidence is a physical measurement, the score and rule checks have nothing to re-run, and CoE depends on the record the lab keeps.

4 **ScientistOne**: Research with Verifiability

We now describe **ScientistOne**, an end-to-end autonomous research system whose three stages are built to meet the CoE requirements: each module records where its outputs came from, so that claims can later be checked against their evidence (Figure 2). **ScientistOne** takes as input a

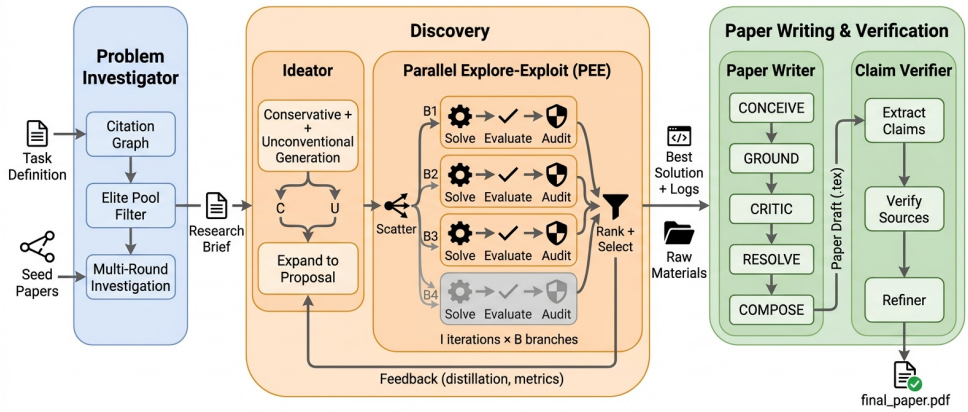

Figure 2: **`ScientistOne`** pipeline. Stage 1 grounds the literature via retrieved PDFs. Stage 2 explores and evaluates solutions across parallel branches. Stage 3 writes and verifies the paper.

research task, specified by a small set of seed papers that scope the topic and a benchmark with its evaluation pipeline, which scores candidate solutions on that task. It produces a complete research artifact: a written paper together with the code, execution logs, and evaluator outputs that back its claims. Pipeline details, models and prompts are in Appendices K and K.1.

4.1 Stage 1: Literature Grounding

The Problem Investigator (PI) collects the references before anything is written: starting from seed papers, it builds a citation graph through scholarly database queries, reads up to 100 full-text PDFs per topic, and produces a structured research brief.

4.2 Stage 2: Solution Discovery

The Ideator generates candidate approaches based on the PI brief, scores them on novelty and feasibility, and distributes the top-ranked proposals across parallel branches of the *Parallel Explore-Exploit* (PEE) orchestrator. Each branch runs an isolated cycle: an Implementer agent iterates up to four evaluated versions per node, with a task-specific evaluator scoring each submission. At each iteration, the top two branches are retained, and the remaining slots are filled with new branches derived from these top performers via fresh ideation. After five iterations across five branches, 25 nodes and about 100 evaluated solutions in every reported run, a best-run selector drops solutions flagged for specification violations, keeps the highest-scoring one, and runs ablations on it.

4.3 Stage 3: Paper Writing & Verification

Paper Writer. The Paper Writer writes the paper in five stages. Conceive writes a structured draft in which every factual claim carries a tag naming the file or log it comes from. Ground mechanically checks that every tagged claim matches its artifact, and Critic reads the whole draft for contradictions, overclaims and missing comparisons. Resolve rewrites the representation to fix both kinds of findings, and Compose turns the checked draft into LaTeX one section at a time.

Claim Verifier and Refinement. The Claim Verifier checks each claim in the composed draft against its declared evidence source: numerical claims against evaluator logs, citation claims against the cited paper's abstract, with an LLM judging whether the abstract supports the claim, and methodological claims against experimental logs. A refinement pass then rewrites each sentence the Verifier flags to match its evidence, keeping every finding and logging each correction with the original wording. The draft is final only when no blocking violation remains.

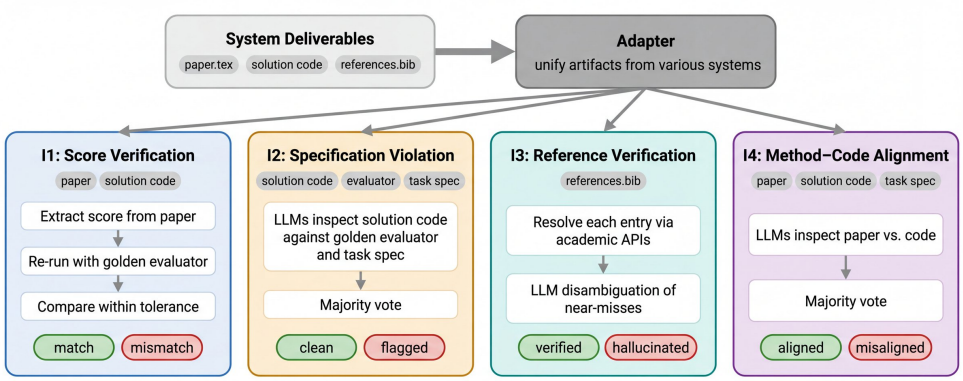

Figure 3: CoE Integrity Audit. An adapter maps each system's deliverables (paper, solution code, bibliography) to a common bundle, and the four checks run on it independently.

5 The CoE Integrity Audit

CoE Integrity Audit is a post-hoc audit that checks whether the claims in a completed paper are supported by the underlying artifacts: code, evaluator outputs, and bibliography (Figure 3, Appendix B). Four checks run independently. I1 and I3 are mechanical: a model only reads the reported score out of the paper or decides whether a bibliography entry and a retrieved paper record are the same paper. I2 and I4 are semantic judgments: an LLM judge runs five times and the majority decides.

Score Verification (I1). The paper's reported score is extracted by an LLM from the TEX and PDF, then compared with the score of re-running the submitted solution on the task's evaluator. A paper passes if the two match within a tolerance set by the evaluator's noise.

Specification Violation (I2). Specification violations occur when solution code breaks task rules, for example by reverse-engineering the evaluator's scoring logic or hardcoding answers for known test cases. LLMs inspect the solution code against the task's evaluator and rules to detect such violations, with a majority vote across independent runs.

Reference Verification (I3). Each bibliography entry is looked up in Semantic Scholar, arXiv, OpenAlex and CrossRef by arXiv ID, DOI and title. When the records are only near matches, an LLM decides whether one is the cited paper. An entry matching no record is a hallucinated reference.

Method-Code Alignment (I4). An LLM reads the paper's method section and the solution code side by side, then judges whether the paper faithfully describes what the code does. Acceptable simplification, such as omitting implementation details, counts as aligned.

6 Experiments

Benchmark. We evaluate on the Automated Design of Research Systems (ADRS) benchmark (Cheng et al., 2025b;a), on five of its research problems from computer systems: **Prism** (LLM-serving model placement across GPUs), **Cloudcast** (cloud network cost optimization), **EPLB** (expert-parallel load balancing for MoE models), **LLM-SQL** (tabular data layout for LLM prefix cache reuse), and **TXN** (transaction scheduling for makespan minimization). We choose ADRS because the leaderboard provides both human expert and recent LLM-agent baselines, and its evaluators are deterministic enough to support Score Verification and Specification Violation detection. ADRS evaluators are known to vary across runs (Cemri et al., 2026; Liu et al., 2026c). We run each evaluator five times and compare against the adaptive tolerance $\max(1\%,\, 3\sigma/|\bar{s}|)$.

Baseline Systems. We evaluate **`ScientistOne`** (§4) and four open-source baselines: **Sakana AI-Scientist v2 (Sakana)** (Yamada et al., 2025) (best-first tree search), **AutoResearchClaw (ARC)** (Liu et al., 2026b) (a 23-stage pipeline with repair and pivot steps), **DeepScientist (DS)** (Weng et al., 2025) (its v1.5.17 release, a skill-based single agent on Codex CLI), and **AI-Researcher (AIR)** (Tang

et al., 2025) (an orchestrated multi-agent system). All five systems, **ScientistOne** included, run on Gemini 3.1 Pro for both solution code generation and paper writing. Details are in Appendix J.

ADRS Adaptation. Each system writes its paper, solution code, logs and bibliography in its own format and directory layout, and we wrote one adapter per system to wire the right artifact to each check. The hard part was choosing the right one among the many solver versions and helper files in each run directory. For each system, we run 3 seeds per task, producing 15 papers per system and 75 papers total. Of the 75 runs, 16 were restarted at least once after failures such as API errors, port conflicts between containers, and LaTeX errors that left no paper PDF. No run was re-attempted to improve its score. Adapting each baseline to ADRS meant patching up to 19 of its source files. That cost is the reason the full audit runs on ADRS alone: a second benchmark would mean repeating the adaptation for every system. We made each adaptation as accurate as we could.

6.1 Audit Results on ADRS

Table 1 gives the results. Every baseline system fails at least one check: in the worst case, 24% of the distinct references do not exist (DeepScientist), 7 of 12 reported scores do not reproduce (AutoResearchClaw), and 3 of 15 method sections match the submitted code (AutoResearchClaw). **ScientistOne** is best or tied for best on all four checks.

Every flagged I1 to I3 case of the four compared systems was checked by a human. Score verification has 12 papers per system instead of 15 because EPLB's score includes a running-time term, so it differs from machine to machine. Reference verification is checked against distinct references, with the raw number of bibliography entries in parentheses. Appendix I gives more details of the results.

Sakana appears in the table but not in the comparison. Its search was built for ML experiments in which the agent designs its own evaluation. On ADRS its solutions import the task evaluator and call it during the search, which the rules forbid, so three of its four columns reflect that mismatch more than the system's own quality. We report its row and compare the other four.

Table 1: CoE Integrity Audit results, 15 papers per system. †Sakana is outside the comparison.

System	Score Verif. ↑	Spec. Violation ↓	Ref. Verif. ↓	Method-Code ↑
Sakana AI-Scientist v2†	4/12	10/15	0/87 (159)	5/15
AutoResearchClaw	5/12	0/15	1/196 (352)	3/15
DeepScientist	11/12	0/15	42/178 (210)	8/15
AI-Researcher	9/12	0/15	21/178 (236)	13/15
ScientistOne	12/12	0/15	0/340 (830)	14/15

Score verification (I1). **ScientistOne** reproduces in 12 of 12 papers, DS in 11, AIR in 9 and ARC in 5. ARC's seven failures are in the submitted code: several solutions crash when re-run because they import helper files that ARC never included, and one does not set a random seed, so its score changes from run to run. DS's one failure treats a cost to be minimized as a score to maximize.

Specification violation (I2). No compared system has a violation. The audit's one flag, an AIR solution that reorders values within each row of the data table, was withdrawn after both blinded raters found the reordering is the task's own design (§6.2), and one DS paper was flagged by two of the five judges, below the majority. The violations in this study are Sakana's.

Reference verification (I3). This check asks whether a reference exists in a scholarly index. In DS 24% of the distinct references do not, in AIR 12%. Both systems write references from the model's memory, which produces plausible entries for papers that were never written. ARC's single case comes from a list of well-known papers written into its source code. **ScientistOne**'s 340 distinct references all exist, because every entry is retrieved as BibTeX from the Semantic Scholar API.

Method-code alignment (I4). **ScientistOne**'s method section matches its code in 14 of 15 papers, AIR's in 13, DS's in 8 and ARC's in 3. ARC's paper-writing stage works from a results summary written by an earlier stage, seven stages after the code was generated, so the description and the code drift apart. **ScientistOne**'s one misaligned paper describes an LLM-guided evolutionary search (the overall discovery process) while the submitted code is a deterministic heuristic with no

LLM calls (the logic of one specific solution version). The Claim Verifier checks numbers, citations and method claims against their sources. It does not check whether the paper's overall methodology matches the code, and it did not flag this. Notably, both human raters considered this misalignment minor and judged the paper aligned (§6.2). This error can be mitigated by adding a check of overall paper–code alignment beyond the current per-claim scope. Appendix E describes one case for each check.

6.2 Human Evaluation of the Audit

The numbers in Table 1 were checked and corrected by humans only where the auto-audit had flagged a case (the positives), and its correctness on the cases it passed (the negatives) is not known. To check the reliability of the audit, and how far humans agree with each other on these audit checks, we ran a blinded human evaluation. Eight raters (seven authors of this paper with PhDs and one doctoral student, all in computer science or a related field) re-judged all 60 papers of the four compared systems from the paper, the code, the task and the evaluator, two raters per paper with a coding agent to help read the code, without seeing which system wrote a paper or what the audit judged.

Table 2: Blinded human evaluation of the auto-audit on the 60 papers of the four compared systems. Denominators short of 60: I1 has 48 papers because EPLB's score does not reproduce across machines. The Raters agree column drops papers where a rater reported unsure. The Auto-audit correct column is computed only on the papers the two raters agreed on (denominator), and counts those where the audit gave the same verdict (numerator). Each reference was annotated by one rater, so I3 has no agreement number. Its 312 references are 220 sampled passes and 92 flags.

Check	Judgment	Items	Raters agree	Auto-audit correct
I1	headline score extraction	48 papers	48/48	48/48
I2	specification violation	60 papers	55/59	54/55
I3	reference is a real work	312 references	–	219/220 passed, 66/92 flagged
I4	method–code alignment	60 papers	50/56 (κ 0.76)	46/50

From Table 2 we observe that, overall, the auto-audit is accurate. On method–code alignment, the agreement between the audit and the raters is about as good as the agreement between the raters themselves (92% against 89%), with precision 83% and recall 94% for detecting misalignment. On reference verification, 219 of the 220 references it had passed are real works, nine of them under a rewritten title (Appendix A), and of the 92 it had flagged (positives), 66 are fabricated and 26 real. In all 48 papers eligible for the I1 check, the headline score is correctly extracted. On specification violation, no paper was judged a violation by both raters, or by a rater and the audit together. The raters split on four papers where the code departs from the task's rules in a way the evaluator does not check, and one rater counted that as a violation while the other did not. This is the difficulty of I2: it requires resolving the ambiguity in each task's definition and in what a solution may do. The one paper the audit flagged is a false positive, judged clean by both raters, because the per-row column reordering it objected to is what the task allows (AIR on LLM-SQL). Its ten flags on Sakana, solutions that import the task evaluator, are right and were not evaluated by humans here.

The errors show where the audit's judgment differs from a human's. On method–code alignment it flagged three papers whose method section is implemented in the code. In each, it objected to a claim made only in the abstract or to a second algorithm in the same file. It passed one paper whose described mechanism the code does not run. On reference verification, the one fabricated reference it passed lists two real authors under a title they never wrote. Its LLM step found a different paper by the same two authors from the same year and accepted that as the match. None of these errors changes how the systems rank in Table 1.

Two improvements follow from its errors: borderline alignment calls should go to a human reader, and the I2 judge should see the task's reference solution. ADRS is also an easy case for an audit: one file per solution and a deterministic evaluator per task. On a software or ML project with many files and no single score, all four checks become harder, and the two semantic checks the most.

Table 3: Solution discovery on ADRS: best of three seeds, each solution re-run on the task's evaluator, in the task's native units (**Dir.**). Human, AdaEvolve and EvoX are quoted from their publications. * marks the Gemini 3.0 Pro backbone.

Task	Dir.	Human	AdaEvo*	EvoX*	Sakana	ARC	AIR	DS	`ScientistOne`	`ScientistOne`*
Prism	↑	21.89	**26.26**	**26.26**	26.00	26.25	**26.26**	**26.26**	**26.26**	**26.26**
Cloudcast	↓	626.24	637.10	623.69	627.11	690.37	734.28	620.09	618.08	**618.07**
EPLB	↑	0.1265	0.1450	0.1453	n/a	0.1266	0.1449	0.1284	**0.1459**	0.1444
LLM-SQL	↑	0.6920	**0.7520**	0.7300	0.7298	0.6757	0.7148	0.7307	0.7222	0.7109
TXN	↑	2724.8	4310	4310	3891	3247	**4311**	4286	3906	3861

6.3 Solution Discovery Performance

Every system we ran matches or exceeds the human expert score on at least three of the five ADRS tasks, and **`ScientistOne`** and DS outperform the human solution on all five (best of three seeds, each solution re-run on the task's evaluator, Table 3). **`ScientistOne`** trails the best system on LLM-SQL and TXN, and with more test-time compute the gap closes on TXN (4348) and narrows on LLM-SQL (0.7316, Appendix D).

6.4 Automated Reviewer Scores

ScholarPeer (Goyal et al., 2026), an automated reviewer on the same backbone every system used to write, accepts 6/15 **`ScientistOne`** papers against 2/15 for the best baseline, AIR (Table 4). It scores Clarity highest for every system and Soundness lowest for four of the five, and its most damaging finding in rejected papers is a claim that contradicts the paper's own data. We read it as a coarse signal.

Table 4: ScholarPeer review scores (1–4 scales, Overall 1–10) and accept decisions, averaged over the 15 papers per system (3 seeds $\times$ 5 tasks).

System	Soundness	Originality	Quality	Clarity	Overall	#Accept
Sakana AI-Scientist v2	1.5	1.9	1.5	3.1	2.5	0/15
AutoResearchClaw	1.1	2.3	1.1	2.5	1.9	0/15
DeepScientist	1.7	1.7	1.6	3.1	2.5	1/15
AI-Researcher	1.9	2.4	1.9	3.1	3.4	2/15
`ScientistOne`	**2.3**	**2.5**	**2.3**	3.0	**4.5**	**6/15**

6.5 Ablation of the Paper Writer

Four writer modules stand between a finished experiment and the finished paper: the reference library the PI built, reference retrieval (a scholarly lookup for every citation the writer adds), citation repair (a pass that fixes citation keys that do not resolve and deletes the ones it cannot fix), and the Claim Verifier. We switched them off in the combinations of Table 5 and re-wrote the same 15 papers from the same raw materials each time. Two measures can respond to these modules: I3, and claim precision, the share of the writer's claims that check out against the evidence they cite.

Table 5: Paper Writer ablation on the 15 **`ScientistOne`** papers.

PI library	Retrieval	Cit. repair	Claim Verifier	I3 unresolvable ↓	Refs/paper	Claim precision ↑
✓	✓	✓	✓	0.0%	68.6	88.6%
✗	✓	✓	✓	0.0%	12.2	86.1%
✓	✗	✗	✓	59.6%	28.7	95.5%
✓	✗	✓	✓	0.0%	8.7	87.6%
✓	✓	✓	✗	0.1%	65.6	90.6%

With all four modules on, the re-written papers reproduce the zero unresolvable rate of Table 1. Retrieval is what makes I3 zero without deleting references: switching off retrieval and repair together leaves 59.6% of references unresolvable ($p < 0.001$, paired Wilcoxon over papers). Switching off retrieval but keeping repair leaves I3 at zero, because repair deletes every key that fails, and the bibliography shrinks from 69 entries per paper to 9. Switching off the PI library alone keeps I3

Table 6: MLE-Bench and Parameter Golf results. Above or Below Median is relative to the median participant, SOTA is the leaderboard top on 2026-04-27.

Task	Dir.	DS score	DS result	`ScientistOne` score	`ScientistOne` result
3D Object Detection	↑	0.0000	Below Median	0.1763	Gold Medal
AI4Code	↑	0.6964	Below Median	0.8356	Above Median
iMet 2020 FGVC7	↑	0.6804	Silver Medal	0.6791	Silver Medal
RSNA Brain Tumor	↑	0.6377	Gold Medal	0.6518	Gold Medal
iNaturalist 2019	↓	0.2158	Silver Medal	0.2445	Silver Medal
Parameter Golf	↓	Invalid	Over the size limit	1.0600	SOTA, constraints met

at zero but shrinks the bibliography to 12 entries. Switching off the Claim Verifier leaves claim precision within noise (89% against 91%, $p = 0.239$). Its contribution is more practical: it produces a claim-evidence record that links each numerical claim in the paper to the log line it came from, and a reader or a script can check the claims easily afterwards.

6.6 Beyond ADRS: MLE-Bench and Parameter Golf

To see whether the discovery loop works outside ADRS, we ran **`ScientistOne`** unchanged on six more tasks, with DeepScientist, the only baseline above the human solution on all five ADRS tasks, on the same six: five MLE-Bench (Chan et al., 2024) Kaggle competitions from the Medium and High difficulty tiers, and Parameter Golf (OpenAI, 2026), a live competition for the best language model trainable under strict size and compute limits. On MLE-Bench the search graded up to 16 candidate submissions on the test set per run and kept the best (the agent itself tunes on a validation split), and medals are the leaderboard's thresholds applied to that score. Table 6 presents the results. **`ScientistOne`** wins gold on the two High-difficulty tasks, RSNA Brain Tumor and 3D Object Detection, where DeepScientist scores zero on the second, both take silver on iMet 2020 and iNaturalist 2019, and **`ScientistOne`** finishes above the median on AI4Code, where DeepScientist is below it. On Parameter Golf both systems received the same knowledge base of official leaderboard solutions, current to April 27, 2026. **`ScientistOne`** reaches 1.0600 bits per byte, better than the leading score in that snapshot, 1.0611, while DeepScientist's submission exceeds the 16 MB size limit and is invalid. More details are in Appendix F.

7 Discussion and Conclusion

Discovery capability is not the bottleneck. Verification is. A deliberately simple search matches the human expert on every ADRS task, a plain coding agent reaches published state of the art on algorithm design (Liu et al., 2026e). More capability also brings more violations: in our scaling runs (Appendix D), the share of LLM-SQL search nodes that game the metric rises from about 0% at 100 evaluations to about 70% at 500. The audit costs 0.05% of the tokens and 0.19% of the dollars that produced the paper it checks (Appendix K.7), cheap enough to run on every submitted solution.

The adaptation tax. Adapting four systems to one benchmark was the most expensive part of this study, and it affected the results. Sakana is the clearest case: its evaluator imports, its missing figures, its method–code mismatches and three of its eight score failures are all because of the adaptation. The checks are the same for every system, but each system needs its own adapter to find its paper, code, bibliography and evaluator log. A standardized structure for research artifacts, with the solution entry point, the evaluator it was scored with, the environment and the run record in fixed places, would make audits portable and reproduction cheaper. This study was possible only because the baselines release their code and intermediate files, and a closed system that exposes only a report cannot be audited this way.

Papers should carry their own evidence record. The audit rebuilds each evidence chain after the paper is written. **`ScientistOne`** records it while writing: each number carries a tag naming its log line, and each citation is resolved against a scholarly index and checked against the cited abstract (§4.3). A check of that record finds 627 of 639 numerical claims consistent with their evidence (Appendix G.2). We leave two checks to future work: whether a cited paper's full text supports the claim citing it, and whether a paper's conclusions follow from its results.

AI USE STATEMENT

Large language models play two roles in this work. They are its subject: **ScientistOne** and the four baselines are LLM agents, every audited artifact was produced by one of them with no human in the loop beyond the adaptation repairs listed in Appendix J, and CoE Integrity Audit itself uses an LLM for the checks that read a source (specification violation, method–code alignment, and the near-match step of reference verification). The body of the paper describes these in full.

They also assisted the authors. LLM-based coding and writing assistants helped write the system and audit code and the analysis scripts, trace individual audit flags to their causes, discuss the study design, and draft and edit the text. The authors reviewed all of it. Every count in Table 1 and in the per-task appendix table is regenerated from the audit records by a script. Every flagged case on the four compared systems was read by an author, and every sentence was revised by the authors. The raters in the human evaluation, seven authors and one doctoral student, read with a coding assistant and recorded that they did. No synthetic data was generated. We take responsibility for the final content of this work.

ETHICS STATEMENT

Autonomous research systems that produce full papers create opportunities and risks. Structured provenance (CoE) and systematic auditing (CoE Integrity Audit) make integrity failures detectable at a scale manual review cannot match: every number and method claim, and the existence of every cited work, can be traced to its source artifact. The same capability lowers the barrier to producing plausible-looking papers at volume, which could flood review pipelines or spread results that look rigorous and carry errors outside the audit's scope. Our checks make verifiable papers distinguishable from unverifiable ones and stop at structural integrity, short of scientific correctness or novelty. Transparency tools like CoE Integrity Audit should develop alongside generation capability, so that the community can verify AI-generated claims instead of taking them on trust.

REPRODUCIBILITY STATEMENT

The search configuration of **ScientistOne** (iterations, branches, retained top-k, evaluation budget per branch), the backbone model and reasoning settings, and the Ideator prompts, with pointers to every other agent's prompt, are in Appendix K.1. The CoE Integrity Audit procedure, including the judge models, the vote depth, the score tolerance and the reference-resolution sources, is in Appendix B.1, and every cell of Table 1 is opened by task in Appendix I. How each baseline was adapted to ADRS, with the source patches and budgets, is in Appendix J. The token and dollar cost of a discovery run and of one audit is in Appendix K.7. The audit's per-vote records are part of its output. The generated papers of all five systems are included as supplementary material. Every solution was re-run on one canonical evaluator per task. The evaluator copies the systems called during their own runs were not pinned to it (Appendix H.1 notes one mismatch). The design of the human evaluation, its per-check counts and the rater agreement are in Appendix A.

REFERENCES

Tim Baumgärtner and Iryna Gurevych. SciCoQA: Quality assurance for scientific paper-code alignment. *arXiv preprint arXiv:2601.12910*, 2026.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, et al. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-Bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yu-Fei He, Jiaying Wu, Yi-Bo Li, Yue Liu, and Bryan Hooi. MLR-Bench: Evaluating AI agents on open-ended machine learning research. *arXiv preprint arXiv:2505.19955*, 2025a.

Tingting Chen, Srinivas Anumasa, Beibei Lin, Vedant Shah, Anirudh Goyal, and Dianbo Liu. Auto-Bench: An automated benchmark for scientific discovery in LLMs. *arXiv preprint arXiv:2502.15224*, 2025b.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Shubham Agarwal, Mert Cemri, Bowen Wang, Alexander Krentsel, Tian Xia, Jongseok Park, et al. Let the barbarians in: How AI can accelerate systems performance research. *arXiv preprint arXiv:2512.14806*, 2025a.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, et al. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025b.

Palash Goyal, Mihir Parmar, Yiwen Song, Hamid Palangi, Tomas Pfister, and Jinsung Yoon. ScholarPeer: A context-aware multi-agent framework for automated peer review. *arXiv preprint arXiv:2601.22638*, 2026.

Theo Härder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. Curie: Toward rigorous and automated scientific experimentation with AI agents. *arXiv preprint arXiv:2502.16069*, 2025a.

Patrick Tser Jern Kon, Jiachen Liu, Xinyi Zhu, Qiuyi Ding, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, et al. EXP-Bench: Can AI conduct AI research experiments? *arXiv preprint arXiv:2505.24785*, 2025b.

Jiachen Liu, Jiaxin Pei, Jintao Huang, Chenglei Si, Ao Qu, Xiangru Tang, Runyu Lu, Lichang Chen, Xiaoyan Bai, Haizhong Zheng, Carl Chen, Zhiyang Chen, Haojie Ye, Yujuan Fu, Zexue He, Zijian Jin, Zhenyu Zhang, Shangquan Sun, Maestro Harmon, John Dianzhuo Wang, Jianqiao Zeng, Jiachen Sun, Mingyuan Wu, Baoyu Zhou, Chenyu You, Shijian Lu, Yiming Qiu, Fan Lai, Yuan Yuan, Yao Li, Junyuan Hong, Ruihao Zhu, Beidi Chen, Alex Pentland, Ang Chen, Mosharaf Chowdhury, and Zechen Zhang. The last human-written paper: Agent-native research artifacts. *arXiv preprint arXiv:2604.24658*, 2026a.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Congyu Zhang, et al. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration. *arXiv preprint arXiv:2605.20025*, 2026b.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, et al. EvoX: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026c.

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Alexandros G. Dimakis, Matei Zaharia, and Ion Stoica. Skydiscover: A flexible framework for ai-driven scientific and algorithmic discovery, 2026d. URL `https://skydiscover-ai.github.io/blog.html`.

Tengxiao Liu, Yuqing Yang, Xi Ye, and Danqi Chen. Can coding agents optimize algorithms autonomously? `https://tengxiaoliu.github.io/autoevolver/`, 2026e. Blog post, March 23, 2026.

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.

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, et al. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. Parameter golf: OpenAI model craft challenge. https://github.com/openai/parameter-golf, 2026.

Razeen A Rasheed, Somnath Banerjee, Animesh Mukherjee, and Rima Hazra. From fluent to verifiable: Claim-level auditability for deep research agents. *arXiv preprint arXiv:2602.13855*, 2026.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, 2025.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-Researcher: Autonomous scientific innovation. *arXiv preprint arXiv:2505.18705*, 2025.

Shuai Wang, Xinyuan Tian, Pangpang Liu, and Yize Zhao. Workflow closure is not scientific closure in auto-research systems. *arXiv preprint arXiv:2605.26200*, 2026.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. Deepscientist: Advancing frontier-pushing scientific findings progressively. *arXiv preprint arXiv:2509.26603*, 2025.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Zhengxin Zhang, Ning Wang, Sainyam Galhotra, and Claire Cardie. How far are we from true auto-research? *arXiv preprint arXiv:2605.19156*, 2026.

Yue Zhao, Mengyuan Li, Ruolin Li, Prince Zizhuang Wang, Shuli Jiang, Linsey Pang, Xiongye Xiao, and Xiyang Hu. CatchBench: When can an agent failure be caught? *arXiv preprint arXiv:2608.22808*, 2026.

APPENDIX CONTENTS

A Human Evaluation of the Audit **14**

B CoE Audit Details **15**

B.1 Audit Procedure and Reproducibility 15

C Paper Quality Statistics **16**

D Solution Discovery **16**

D.1 Solution Discovery Performance 16

D.2 Search Scaling 17

E Case Studies **18**

F MLE-Bench and Parameter Golf Evaluation **18**

F.1 The Parameter Golf Solution 19

G Paper Writer Evaluation Details **20**

G.1 Automated Reviewer Scores 20

G.2 Native Claim Provenance 20

G.3 Ablation: Full Findings 20

H Failure Cases per Audit Metric **21**

H.1 I1: Score Verification Errors 21

H.2 I2: Specification Violation Analysis 23

H.3 I3: Hallucinated References 24

I Every Check by System and Task **27**

J Baseline Adaptation Details **27**

K System Implementation Details **29**

K.1 Search Configuration 29

K.2 Problem Investigator 31

K.3 Implementer 32

K.4 Ablation 32

K.5 Paper Writer 32

K.6 Claim Verifier 32

K.7 Cost and Runtime 33

A HUMAN EVALUATION OF THE AUDIT

Design. The 60 papers of the four compared systems (15 each) were packed into five task bundles of 12 with system and seed removed and the paper order shuffled. Each bundle went to two raters, who judged every paper in it on I2 and I4 from the paper, the solution code, the task specification and the evaluator, and did not see the audit's verdicts. The raters are seven authors of this paper, each holding a PhD in computer science or a related field, and one doctoral student in computer science who is not an author. Each recorded whether a coding agent read with them and whether they took its conclusion. Two authors also judged I1 on the 48 papers eligible for the check (whether the number the audit extracted is the paper's headline result). The I3 sample is described below. Labels were scored against the audit's verdicts. A paper counts as agreeing with the audit when both raters gave the same label. Papers the raters split on are reported separately.

Reference sample. The resolver verified 1,535 references across the four systems: 190 by DOI or arXiv identifier, 1,225 by an exact title match, 96 by a fuzzy title match or the disambiguation model, and 24 through a secondary lookup path. The sample took all 120 of the last two groups, where a fabricated entry is most likely to be missed, a task-stratified 100 of the exact-title matches, and none of the identifier matches. Each reference was checked by one author from its title and bib entry with a web search, and labeled found, unsure or not found, with a reason from a fixed list for the last two. The references not labeled found were then adjudicated by the authors against the bibliographic record, under one rule: an entry is real if the annotator can identify, from its title, a paper that exists, allowing misspellings, truncations and changed subtitles, and fabricated otherwise. The 92 flagged entries (Appendix B.1) were reviewed under the same rule. Where a changed subtitle ends and a rewritten title begins is not a reliable judgment, since a title can change between versions of a paper or be invented. The uncertain cases are entries with a rewritten title whose authors, venue and year belong to a real paper. Ten of the 64 distinct flagged works (12 of the 66 entries, Appendix H.3) are such entries, and they are counted as fabricated. Nine of the 220 passed references are such entries, and they are counted as real. Counting all nineteen as real would change ARC to 0/196, AIR to 14/178 and DS to 40/178 in Table 1, counting all nineteen as fabricated would change the passed sample from 219 to 210 of 220, and in neither case does a system overtake another (ARC would tie **`ScientistOne`** at zero). The resolver flagged 92 entries. The authors checked each of them before the audit was reported: 66 are fabricated and 26 are real. Table 1 counts distinct works, and the 66 entries correspond to 64, because ARC's one fabricated work is cited by three of its papers.

Score verification (I1). The number the audit extracted is the paper's headline result in all 48 papers eligible for the check, and the two raters agreed on all 48. For the two papers with no headline number, the audit extracted none and the raters confirmed there is none.

Specification violation (I2). Both raters judged 55 of 59 papers clean and split on four (ARC 3, DS 1). No paper was judged a violation by both. The one paper the audit flagged (AIR llm_sql seed 1) was judged clean by both raters, and the flag is wrong: the per-row column reordering it objected to is the task's own design.

Reference verification (I3). The raters marked 17 of the 220 sampled references not found and 9 unsure, 22 of them among the 96 fuzzy or model-disambiguated matches, 2 among the 24 secondary lookups and 2 among the 100 exact-title matches. (These 26 are a different set from the 26 flagged entries relabeled as real in Appendix B.1.) Checked against the bibliographic record, 25 are real works with a wrong title, co-author or identifier in the entry, and the resolver had matched each to the work intended. One, a DeepScientist entry with real authors and an invented title, is fabricated: the disambiguation step accepted a different 1989 paper by the same authors. The resolver therefore passed one fabricated reference among the 220, in the fuzzy stratum, and none among the exact-title matches. All 93 **`ScientistOne`** references in the sample are real works.

Method–code alignment (I4). The auto-audit agrees with the raters' shared judgment on 46 of 50 papers, and the raters agree with each other on 50 of 56. It judged three papers misaligned that the raters judged aligned (**`ScientistOne`** cloudcast seed 1, AIR cloudcast seed 1, ARC llm_sql seed 2). In each the method section is implemented and the audit objected to something outside it: an evolutionary-loop framing of code that is a fixed sequence of solvers, a routing step claimed in the abstract but absent from the code, and a partitioner in the entry class of a file whose method is in a second class. It passed one paper the raters judged misaligned (DS eplb seed 2), where the code

does not run the described mechanism: an exponential moving average whose history term the code multiplies by zero. None of the five judges flagged it. Six papers split between the raters (AIR 2, DS 1, ARC 3), four have one rater unsure, and these ten are left out. Rater agreement was highest when both read with a coding agent (32 pairs, κ 0.80) or the record is missing on one side (15 pairs, κ 0.84), and lowest when one rater took the agent's conclusion (9 pairs, κ 0.50).

Cost. Raters logged 23.8 hours on 118 timed paper judgments, a median of 4 to 16 minutes per paper depending on the rater. Coding agents read alongside the rater on 92 of the 120 paper judgments, gave the conclusion the rater adopted on 13, and went unrecorded on 15.

B CoE Audit Details

B.1 Audit Procedure and Reproducibility

Table 7 summarizes the automation level of each component. Score Verification uses LLM-based extraction to identify the paper's reported score from TEX and PDF files, then compares it against reproduced scores from evaluator re-runs via numeric tolerance. The comparison itself is deterministic. Specification Violation and Method-Code Alignment are LLM-judged, with majority vote over five independent runs to reduce judgment noise. Reference Verification is primarily API-based (Semantic Scholar, arXiv, OpenAlex, CrossRef lookup), with an LLM disambiguation step for title-only matches.

Table 7: CoE Integrity Audit automation level per component.

Component	Automation	Model (if LLM)
Score Verification	LLM extraction + automated comparison	Gemini 3 Flash
Specification Violation	LLM-judged	Gemini 3.1 Pro
Reference Verification	Automated + LLM disambiguation	Gemini 3 Flash
Method-Code Alignment	LLM-judged	Gemini 3.1 Pro

Human review of the flags. Every case the audit flagged on I1, I2 and I3 was reviewed by an author before Table 1 was reported, and a flag the review overturned was removed. The review touched only flagged cases (the blinded evaluation in §6.2 and Appendix A is different: two raters judged every paper, and a sample of passed references, without seeing the audit's verdicts). It corrects the audit's false positives (API resolution failures for real papers, score extraction errors) and cannot find the audit's misses. On I3 it removed 26 of the 92 references the resolver could not confirm, and the reported numerators are therefore lower than the raw check output. The one I2 flag among the compared systems (AIR) was withdrawn after the blinded evaluation in Appendix A judged it clean. The I4 flags were read by an author too. That reading found two false positives, both caused by the audit judging an outdated solution file, and both were fixed by re-running the audit on the right file. No I4 verdict was changed by hand, and the column is the audit's majority vote as it ran. Appendix A measures the audit's accuracy on flagged and passed cases alike, and covers I4 on all 60 papers of the four compared systems.

How the I3 flags were reviewed. The reference column carries the largest correction. This is how it was produced. The resolver could not confirm 92 references: 45 from DS, 44 from AIR, 3 from ARC, and none from **`ScientistOne`** or Sakana. An author looked each one up. 66 are fabricated and 26 are real works the resolver missed. Fourteen of the 26 have the same cause: a `.bbl`-to-`.bib` conversion left the author field empty and put the whole rendered bibliography item into the title field, and the resolver queried that as a title and found nothing. Each of the 92 got a label, real or fabricated, and a one-line reason, recorded in one correction file. The script that builds Table 1 reads that file on top of the raw audit records, and no number in Table 1 was edited by hand.

Two checks say the review was consistent. First, the 14 entries with the broken-conversion form are all among the 26 real ones, and none of the 66 fabricated ones has it. Second, the corpus has 784 distinct titles that resolve somewhere across its 1,787 entries, and none of the 66 fabricated titles is among them. A real work wrongly called fabricated would be likely to appear as a resolvable title elsewhere in a corpus this size, and none does. Neither check can find a fabrication the resolver never flagged. The sample in Appendix A covers that case: one of the 220 it passed is fabricated, and nine

more cite a real work under a rewritten title (a rewritten title can come from a revision of the paper or from a hallucination, and the raters could not tell which).

C Paper Quality Statistics

We report structural statistics of all 75 papers (5 systems $\times$ 3 seeds $\times$ 5 tasks) to characterize the surface-level quality of AI-generated manuscripts. Statistics are extracted automatically from the LaTeX source and compiled PDF of each paper.

Table 8 reports the mean and standard deviation of each metric across the 15 papers per system. ARC papers are the longest (12.7 $\pm$ 1.2 pages, 5,417 $\pm$ 863 words) and most figure-heavy (4.3 $\pm$ 2.0 per paper). **`ScientistOne`** has the largest bibliography pool (55.3 $\pm$ 3.6 entries) and 3.8 $\pm$ 0.4 figures per paper, though only 18.3 $\pm$ 4.5 keys are cited, fewer than AIR's 19.5 $\pm$ 3.9 unique citation keys. AIR is equation-heavy (7.1 $\pm$ 1.8 per paper) and table-heavy (4.9 $\pm$ 1.5) but generates nearly zero figures (0.1 $\pm$ 0.2). Sakana produces the shortest papers (5.8 $\pm$ 0.8 pages, 2,606 $\pm$ 490 words) with zero figures across all 15 papers. DS is similarly brief (6.7 $\pm$ 2.1 pages).

Table 8: Paper quality statistics across five systems (mean $\pm$ std over 15 papers each).

Metric	ARC	DS	AIR	Sakana	`ScientistOne`
Pages	12.7 $\pm$ 1.2	6.7 $\pm$ 2.1	10.4 $\pm$ 1.1	5.8 $\pm$ 0.8	10.2 $\pm$ 0.8
Words	5,417 $\pm$ 863	2,741 $\pm$ 968	5,116 $\pm$ 524	2,606 $\pm$ 490	4,643 $\pm$ 438
Figures	4.3 $\pm$ 2.0	1.3 $\pm$ 0.4	0.1 $\pm$ 0.2	0.0 $\pm$ 0.0	3.8 $\pm$ 0.4
Tables	2.1 $\pm$ 0.7	0.8 $\pm$ 0.7	4.9 $\pm$ 1.5	2.3 $\pm$ 1.1	1.9 $\pm$ 0.2
Equations	1.3 $\pm$ 1.6	1.9 $\pm$ 2.0	7.1 $\pm$ 1.8	2.2 $\pm$ 1.5	3.0 $\pm$ 1.1
Unique cite keys	23.5 $\pm$ 8.7	9.3 $\pm$ 6.9	19.5 $\pm$ 3.9	10.5 $\pm$ 2.5	18.3 $\pm$ 4.5
Bib entries	23.5 $\pm$ 8.7	13.1 $\pm$ 13.2	15.8 $\pm$ 1.9	10.6 $\pm$ 2.6	55.3 $\pm$ 3.6
Sections	6.6 $\pm$ 2.5	6.5 $\pm$ 1.0	5.0 $\pm$ 0.0	6.5 $\pm$ 0.7	5.5 $\pm$ 0.6
Subsections	9.1 $\pm$ 3.9	9.7 $\pm$ 3.4	9.9 $\pm$ 1.7	6.0 $\pm$ 1.1	7.5 $\pm$ 1.4

D Solution Discovery

D.1 Solution Discovery Performance

Table 3 reports best-of-3 scores across seeds on ADRS, following Cemri et al. (2026). Each seed's score comes from re-running the solution that run submitted under the canonical evaluator. We do not quote the score the paper prints, because §6.1 shows that for several systems the printed score does not reproduce.

Sakana's row in Table 3 comes from a separate pass with five evaluator repeats per seed, and its EPLB entry is n/a because the EPLB score includes a wall-clock term that does not reproduce across hardware.

The table is a control for the audit: the systems whose papers we audit solve the tasks, so their audit records are not explained by weak solutions.

On Cloudcast Sakana (627.11), AdaEvolve (637.10), ARC (690.37), and AIR (734.28) fall short of the human baseline, and on LLM-SQL ARC (0.6757) falls short of the human 0.6920. Among the audited systems, **`ScientistOne`** and DS exceed it on every task. Sakana's papers report different numbers (§6.1). With the search configuration fixed, swapping Gemini-3.1-Pro for Gemini-3.0-Pro leaves Prism and Cloudcast unchanged within evaluator noise and costs a little on the other three tasks (EPLB 0.1459 to 0.1444, LLM-SQL 0.7222 to 0.7109, TXN 3906 to 3861).

Hard failures under re-evaluation. The submitted solution produced no usable score on all three canonical re-runs of the discovery evaluation for 3 of ARC's 15 runs and for none of the 45 runs of AIR, DS and **`ScientistOne`**.

D.2 SEARCH SCALING

Table 3 reports **`ScientistOne`** scores from the final iteration of the best-performing branch (1 of 5 parallel branches). Table 9 reports the best score across *all* nodes in the search tree, measuring the discovery module's ceiling under varying tree and budget configurations. Solutions are manually checked and specification-violating ones (metric gaming on LLM-SQL, abusing the Prism scoring formula by failing on subsets of test inputs) are excluded. All scaling configurations report a single seed and cross-seed variance can be substantial (e.g., TXN ranges from 3636 to 3906 across three seeds with the base configuration). These results are therefore directional.

Tree and budget scaling. We evaluate search configurations along three axes: width (parallel branches B), depth (iterations I), and per-node evaluator budget (E, maximum evaluated solution versions per node). Table 9 reports the best score across all nodes for each configuration (single run each). The tree scaling block in Table 9 varies the tree structure (I, B, K) at fixed budget ($E=4$). The budget scaling block varies E while holding the tree fixed at $I=5, B=5, K=2$.

Table 9: Best score across all search tree nodes under different tree and budget configurations (single run each). I=iterations (depth), B=branches (width), K=branches retained per iteration, E=max evaluator calls per node. "Base" is the configuration used for Table 3. **Bold** marks the best score in each column, computed separately for baselines and **`ScientistOne`** configurations.

Config	I	B	K	E	Nodes	Prism↑	Cloud.↓	EPLB↑	LLM-SQL↑	TXN↑
Reference baselines (best-of-3 seeds, selector-based; from Table 3)										
Human			–			21.89	626.24	0.1265	0.6920	2724.8
AdaEvolve			–			**26.26**	637.10	0.1450	**0.7520**	4310
EvoX			–			**26.26**	623.69	**0.1453**	0.7300	4310
Sakana			–			26.00	627.11	n/a	0.7298	3891
ARC			–			26.25	690.37	0.1266	0.6757	3247
AIR			–			**26.26**	734.28	0.1449	0.7148	**4311**
DS			–			**26.26**	**620.09**	0.1284	0.7307	4286
***`ScientistOne`** scaling configurations (1st seed of 3, best across all nodes)*										
Base	5	5	2	4	25	26.26	618.09	0.1287	0.7299	3636
Tree scaling (fixed budget $E=4$)										
No pruning	5	5	5	4	25	26.26	618.08	**0.1461**	0.7092	3663
Wide	5	10	2	4	50	26.36	618.09	0.1369	0.7215	4082
Wider	5	15	2	4	75	26.26	618.08	0.1456	0.7189	4237
Widest	5	20	2	4	100	**26.44**	**618.07**	0.1455	0.7257	4255
Deep	10	5	2	4	50	**26.44**	618.08	0.1460	0.7118	3831
Deep+wide	10	10	4	4	100	26.33	618.09	0.1458	0.7078	4000
Budget scaling (fixed tree $I=5, B=5, K=2$)										
Budget 200	5	5	2	8	25	26.26	**618.07**	0.1284	0.7256	**4348**
Budget 500	5	5	2	20	25	26.34	**618.07**	0.1456	**0.7316**	4237

Three patterns emerge from tree scaling. First, TXN scales monotonically with width: scores increase steadily from 3636 (base, $B=5$) through 4082 ($B=10$), 4237 ($B=15$), and 4255 ($B=20$), a 17% improvement at the widest configuration and approaching AdaEvolve (4310). Second, EPLB benefits from scale but saturates early: most non-base configurations reach $\sim$0.146 (a 13% improvement over the base's 0.129), with the exception of Wide ($B=10$, $K=2$) at 0.137. Third, Cloudcast, LLM-SQL, and Prism largely saturate: all configurations converge to similar scores on these three tasks regardless of tree shape, suggesting a narrow basin of high-performing solutions that the default search finds quickly.

Taken together, width (more independent branches) is the most efficient scaling axis for tasks with diverse solution strategies. The widest tree ($B=20$, 100 nodes, $E=4$) matches or exceeds the highest per-node budget configuration ($E=20$, 25 nodes) on 3 of 5 tasks, despite using 5$\times$ fewer evaluator calls per node. Budget scaling shows gains on TXN (budget 200 reaches 4348, +20%) but saturates quickly. Budget 500 does not improve further on TXN, and the remaining tasks are flat across budget levels except EPLB, which rises from 0.1287 to 0.1456 at budget 500, and Prism, from 26.26 to 26.34. Depth and budget produce diminishing returns once the search covers the main algorithmic strategies.

Specification violations at higher budgets. Increasing per-node budget amplifies specification violation risk. At budget 100, no specification violations are observed on Prism. At budget 200 and 500, 2–8% of nodes converge to solutions that exploit the scoring formula without solving the task. LLM-SQL shows a similar trend: the fraction of nodes flagged for metric gaming by the post-hoc auditor grows from $\sim$0% at budget 100 to $\sim$50% at budget 200 and $\sim$70% at budget 500. In contrast, wider trees at budget 100 show lower violation rates despite producing more total nodes, because each node has fewer iterations to discover and refine exploitative patterns.

E CASE STUDIES

Four cases from the 75-paper audit.

Case 1: Six orders of magnitude (ARC, LLM-SQL, seed 2). The paper introduces "SCOR," a static column-ordering routine, and reports a combined score of 1,538,006.69 on a benchmark whose scoring metric uses a $[0, 1]$ scale. The number is the sum of squared prefix-hit lengths across datasets, an internal metric the system presented as the ADRS score. The paper is internally coherent: it defines its own evaluation protocol, runs controlled comparisons against a baseline (scoring 1,537,927.99), and draws reasonable conclusions within that framing. Score Verification catches this: the canonical evaluator re-run crashes (the submitted code fails to produce a valid solution), making the entire evidence chain from score to evaluator unresolvable.

Case 2: A bibliography from model memory (AIR, PRISM, seed 1). This paper's bibliography contains 15 references. Reference Verification finds that 3 of those references are hallucinated: we could not find matching publications in Semantic Scholar, arXiv, or other scholarly databases. The case is typical: AIR and DS produce hallucinated references at rates of 12% and 24% of distinct cited works (Table 1), against 0% for systems with structured retrieval pipelines.

Case 3: A near miss that is not a violation (DS, LLM-SQL, seed 1). Two of five I2 judges flagged this paper, below the majority threshold. The code sorts columns differently per row-group block and concatenates the blocks, so the column order differs from row group to row group. The evaluator checks row counts and total character counts and not the column order, and the task's own reference solution reorders columns per row group in the same way, so the pattern is within the task's rules. The same pattern appears in AIR seed 1, where the audit's majority flag was withdrawn after both blinded raters judged it clean (§6.2), and in **`ScientistOne`** seed 2, where one of five judges flagged it. The judges' disagreement on all three papers is the I2 ambiguity that §6.2 describes.

Case 4: Near-correct score, fictional algorithm (ARC, TXN, seed 1). This paper nearly passes Score Verification: the reported score of 3,311 is within 3% of the canonical evaluator re-run mean (3,214), just outside the adaptive tolerance threshold. Method-code alignment finds that the code does not implement the method the paper describes. The paper introduces "STAR," a system built on bitwise integer encoding for conflict detection, an $O(1)$ surrogate cost model, and equidistant placement of high-contention anchor transactions. The submitted code implements none of these: it uses standard Python sets for conflict tracking, calls the full simulator on every iteration (no surrogate), and clusters read-heavy keys sequentially where the paper describes distributed write-heavy anchors. A paper like this can pass score verification. The solution works and the paper describes a different algorithm, and a reader cannot reproduce the method from the paper.

F MLE-BENCH AND PARAMETER GOLF EVALUATION

Table 10: Mapping between MLE-Bench task names used in our evaluation and their corresponding official task IDs.

Task Name	Task ID
3D Object Detection	`3d-object-detection-for-autonomous-vehicles`
AI4Code	`AI4Code`
iMet 2020 FGVC7	`imet-2020-fgvc7`
RSNA Brain Tumor	`rsna-miccai-brain-tumor-radiogenomic-classification`
iNaturalist 2019 FGVC6	`inaturalist-2019-fgvc6`

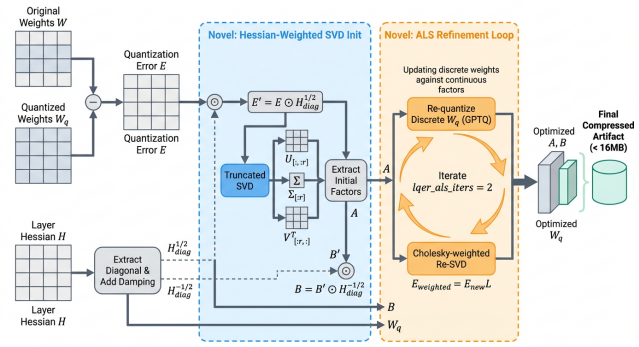

Figure 4: The Parameter Golf solution generated by **`ScientistOne`**: a Hessian-diagonal-weighted SVD initialization and a GPTQ-driven alternating-least-squares refinement loop in the quantization block.

We evaluate **`ScientistOne`** on MLE-Bench and Parameter Golf. Details regarding the benchmarks and our evaluation setup are provided below.

MLE-Bench. We chose five MLE-Bench (Chan et al., 2024) Kaggle competitions in the Medium and High difficulty tiers: AI4Code, iMet 2020 FGVC7 and iNaturalist 2019 FGVC6 (Medium), 3D Object Detection and RSNA Brain Tumor (High). Table 10 maps the names used here to the official task IDs. The agent receives the task description and a code execution environment on one node with 8 H100 GPUs, 192 CPU cores and 1 TB of RAM. It can call a submission-format validator without limit. The grading server scores each search node's final submission on the test data, up to 16 per run, and the search keeps the best. The official protocol scores one final submission. We allow the 16 gradings because Kaggle participants also submit to a public leaderboard during development, and because the paper has to report the final test metric. The agent iterates on validation metrics and never sees a test score.

Parameter Golf. We also evaluate **`ScientistOne`** on Parameter Golf (OpenAI, 2026), a live competition on language-model training. It requires participants to train the highest-performing language model under strict constraints: the final artifact must fit within a 16MB size limit and the training process must complete in under 10 minutes on an 8xH100 system. Performance is evaluated by the compression rate, measured in tokenizer-agnostic bits per byte (BPB) on the FineWeb validation set. We provide the agent with a sandbox tool to execute solution code to obtain evaluation metrics. Additionally, the agent is provided with a knowledge base containing official leaderboard solutions up to a cutoff date of April 27, 2026. As of this date, the state-of-the-art (SOTA) score was 1.0611, achieved by a solution titled "BOS-Fixed SmearGate + LQER + SparseAttnGate + 9-Hparam Stack". The agent is asked to improve on the leaderboard solutions.

We did not run the audit on the MLE-Bench and Parameter Golf papers. Reference verification would run as is, since it reads only the bibliography, and score verification would need one grading-server call per paper. Specification violation and method–code alignment would not: both read the one solution file that produced the score, and on these tasks the score comes from a repository of several files and a training run. Extending those two checks to repositories is the adaptation this study did not do.

F.1 The Parameter Golf Solution

Both systems receive a knowledge base of official leaderboard solutions up to April 27, 2026, when the leading score was 1.0611. The leaderboard has since moved past it.

Solution novelty on Parameter Golf. Both systems started from the same prior-art reference and reached similar numbers by different routes. **`ScientistOne`** changed the quantization block: a Hessian-diagonal-weighted SVD initialization and an alternating-least-squares (ALS) refinement loop that uses GPTQ and a Cholesky-weighted truncated SVD (Figure 4). Internal ablations attribute most of the gain to the ALS loop. DeepScientist changed no algorithm. Its edits were environment

and portability adjustments, its score matched the reference, and its submission exceeded the 16MB artifact limit.

G Paper Writer Evaluation Details

G.1 Automated Reviewer Scores

Accept rates. **`ScientistOne`** reaches a 40% accept rate (6/15), three times the best baseline (AIR, 13%), and best-of-3 selection reaches an overall rating of 6.6 with 4 of 5 tasks accepted. Every system matches or exceeds the human solution on at least three of the five tasks (Table 3), so the gap opens after the discovery stage. The most damaging failure in rejected papers is a claim that contradicts the paper's own data, such as "sub-millisecond latency" when the results table reports 7.9 ms. The Claim Verifier reads for exactly this, and the ablation in §6.5 leaves open how much of the gap it accounts for.

Research soundness is the bottleneck. Across all systems, Clarity is the highest-scoring dimension (2.5–3.1) and Soundness the lowest or tied lowest (1.1–2.3): the papers read well and do not withstand methodological scrutiny. The reviewer's two most frequent complaints are missing comparisons against published baselines and proxy-only evaluation without end-to-end system measurements. While **`ScientistOne`**'s Problem Investigator retrieves related work and identifies candidate baselines, the resulting comparisons do not yet meet the depth that ScholarPeer expects (e.g., re-implementing a SOTA method and reporting head-to-head numbers). **`ScientistOne`** also shows high seed variance (EPLB overall scores of 1, 3 and 8 across three seeds of the same task). In the rejected runs the writer makes claims outside the Claim Verifier's current coverage, for example qualitative framing such as "near-optimal" that no number can falsify. The accepted runs make calibrated claims from the same underlying data, which suggests that extending verification to qualitative claims would reduce the variance.

G.2 Native Claim Provenance

The forensic audit (§6.1) runs the same four checks, judge prompts and vote thresholds on every system, using only the submitted artifacts. The audit tool takes artifact paths as arguments and has no per-system branch. Only the adapter that locates each system's deliverables (paper, solution code, bibliography, evaluator log) in its own output layout varies across systems. For **`ScientistOne`**, which emits structured provenance chains at write time, we run one additional *native* check on those chains: numerical CPR (Claim Provenance Rate). No other system in the evaluation produces the provenance records it needs.

Numerical CPR measures whether quantitative claims in the paper trace to experimental evidence. During paper generation, the writer annotates each sentence containing a number with a {`source: "experimental_log.md:N"`} tag linking it to a specific log line. The claim verifier (`check_sources`) extracts the number from the sentence and the referenced log line, then checks whether they match within a 5% relative tolerance.

Across 15 papers (3 seeds $\times$ 5 tasks), the verifier extracts 639 numerical claims. Of these, 627 pass (98.1%). Most of the 12 failures are false positives of the extraction heuristic: hardware constants parsed as experimental claims ("80GB GPU" matched against an unrelated log line), LaTeX subscripts extracted as numbers ($S_{k-1} \to -1.0$), and hyperparameter values in the methodology section. Manual inspection finds at most 2–4 genuine mismatches among the 12, for a corrected numerical CPR near 99%. The paper-writing ablation in §6.5 finds that the writer's claim precision does not separate conditions, so CPR describes the artifact's traceability and has little power as a discriminating score.

G.3 Ablation: Full Findings

Which checks can respond. Section 6.1 attributes **`ScientistOne`**'s reference result to retrieval. To test those attributions we re-wrote the 15 Table 1 papers with one paper-writing module disabled at a time, from identical raw materials, so every condition is paired by task and seed and only the writer's flags differ. Three of the four writer modules act between the finished experiment and the

finished paper: reference retrieval (the seed library the PI built during ideation, a baseline Semantic Scholar lookup, and a literature search), citation repair (backfill, reconciliation, and deletion of any citation key that does not resolve), and the Claim Verifier.

A check can respond to this ablation only if the ablated module feeds the evidence it reads. I2 reads solution code and the evaluator and never opens the paper, and I1's claimed score traces to the raw materials, which are identical across conditions, so neither can move. I4 sits within one paper of ceiling in every condition and the full system moves by one paper against its own replicates, so it has no resolution at this sample size. That leaves I3 and claim precision, the fraction of the writer's annotated claims that bind to their evidence. Claim precision is measured here by the July 2026 build of the writer, which re-wrote the papers, a different instrument from the numerical CPR of §G.2.

Retrieval carries the reference result. Removing retrieval together with citation repair takes unresolvable references from 0.0% to 59.6% (paired exact Wilcoxon per paper, mean paired difference +60.3 points, $p < 0.001$), the only contrast in the table that reaches significance. The contrast reproduces on an independently rebuilt pipeline at 55.5% (+56.1 points, $p < 0.001$), with the control again at 0.0%. The ablated rate is a floor: a reference whose lookup hits a technical error counts as not hallucinated, and such errors are most frequent in the conditions with the most fabrication.

Citation repair can hide the absence of retrieval. With retrieval off and citation repair on, unresolvable references stay at 0.0% while the bibliography shrinks to 8.7 entries per paper: every key that did not resolve was deleted before the paper compiled. A resolvability check on its own cannot tell a system that retrieves its references from one that deletes the ones it invented, so I3 has to be read together with bibliography size, and Table 1 prints the entry count next to every ratio for that reason.

The PI's library sets coverage, and citation repair sets correctness. Disabling only the PI's seed reference library leaves the unresolvable rate at 0.0% on all 13 papers and leaves claim precision where it was. What collapses is coverage: mean bibliography size falls from 68.6 entries to 12.2, every paper drops, and the two distributions do not overlap (the smallest bibliography with the library has 62 entries and the largest without has 23). On two of the fifteen runs the remaining channels returned 0 and 1 usable entries, against 67 and 66 in the control, and the writer's bibliography gate stopped the run. We report them as refusals and keep them out of the rate.

The Claim Verifier does not move claim precision. Claim precision is 88.6% with the verifier and 90.6% without ($p = 0.239$), and on the second pipeline 91.2% against 91.7% ($p = 0.519$), both differences slightly favoring the ablated condition. The flag took effect, since the claim-evidence file is present for all 15 control papers and absent for all 15 ablated ones, so this is a real null. It is also an underpowered one: with 7 to 17 bindable claims per paper, one claim moves a paper's rate by 6 to 14 points, more than any contrast in the column. On these papers the writer makes few unbindable numerical claims with or without the verifier. The verifier's contribution is the claim-evidence artifact itself, which is what makes a paper auditable, and this ablation does not measure how often it changes a claim.

The PI is a provenance module. A separate ablation at the discovery stage removes the PI brief from the search and leaves ADRS scores where they were. Three-seed means over these runs, which are not the runs behind Table 3, with and without the brief are 26.26 and 26.31 on Prism, 618.08 and 618.09 on Cloudcast, 0.1403 and 0.1402 on EPLB, 0.7252 and 0.7174 on LLM-SQL, and 3753 and 4110 on TXN, where the gap is about 1.2 times TXN's cross-seed standard deviation of about 287. Removing the PI costs nothing on the score and costs the paper most of its literature, including two runs that could not be written at all.

H Failure Cases per Audit Metric

H.1 I1: Score Verification Errors

We reviewed every I1 (Score Verification) failure flagged across the five systems and categorize the 23 flags into five classes (Table 11). Sakana's eight flags were not part of the recorded human review, so we traced all eight from the run's own files: five are the paper's doing and three arise

in re-evaluation. Table 1 therefore reports Sakana's I1 count and leaves it out of the cross-system comparison.[1]

Table 11: I1 flags by class. Three of Sakana's eight arise in re-evaluation.

Category	Definition	Count	%
`value_mismatch`	Paper headline number differs from evaluator rerun beyond tolerance, with the same metric and scale.	14	61%
`cross_stage_cherry_pick`	Writeup LLM selects a score from a different search-tree stage (e.g. ablation) than the node whose code is submitted.	4	17%
`paper_score_unavailable`	Paper contains no machine-readable headline number for the auditor to verify against.	2	9%
`evaluator_error`	Submitted artifact cannot be re-evaluated within the budget (timeout or crash on the agent's own solution file).	1	4%
`metric_mismatch`	Paper and code report different metrics, scales, or optimization directions, so the numbers are not directly comparable.	2	9%

`value_mismatch` (n=14). The paper and the code agree on what is being measured but disagree on the value. Most (10/14) of these gaps are within 5% of the paper-reported number, small enough to arise from unreported seed variance, but uniformly biased towards a better-than-rerun headline. Among the four baseline systems, the two largest gaps are qualitatively different errors. One is a fabricated metric direction in DS `cloudcast` seed-2 (26.7%), where the paper relabels a cost metric as "utility" and inverts the optimization direction so that a worse-than-baseline number reads as an improvement. The other is a $2\times$ mismatch in ARC `prism` seed-3 between the paper headline (12.74) and what the submitted code produces (26.24). Sakana ASv2 contributes three `value_mismatch` cases. In `llm_sql` seed-1 the paper reports the best value of an in-file hyperparameter sweep (0.7297 in the node's record), and the submitted file leaves the last setting tried in place, so it re-runs to 0.676 in its own directory and 0.672 in ours. The other two are not misreports. `prism` seed-0 comes from the working directory: the solution tunes its hyperparameters by evaluating `runfile.py` in the current directory, a file present in the BFTS run and absent under standalone re-evaluation, so the sweep fails and the solution falls back to defaults. Re-running the file gives 26.256 with `runfile.py` present and 22.339 without it, against a paper claim of 26.25. `cloudcast` seed-2 reports the 639.13 its node recorded, and the solution, an unseeded multi-start search under a time budget, re-runs to 627.7, 1.8% off and outside the 1% tolerance.

`cross_stage_cherry_pick` (n=4). All four cases come from Sakana ASv2 and reflect a boundary mismatch in the BFTS pipeline: the node the writeup describes comes from the search stages, but the writeup LLM receives summaries from all four stages, ablation included, and picks the most favorable score from the entire pool. For example, in `prism` seed-1 the selected node scores 22.79, but the paper reports 25.39, a number traced to ablation node 6 ("Ablate KVPR-Aware Initialization") in the ablation summary. The same pattern appears in `cloudcast` seed-0 (+56%), `prism` seed-2 (-4.7%, paper under-reports), and `txn_scheduling` seed-2 (+17%).

`paper_score_unavailable` (n=2). The agent's writeup contains no machine-readable quantitative result for the headline metric.

`evaluator_error` (n=1). The artifact the agent declares as its solution cannot be re-evaluated end-to-end within the budget. The single case is ARC `llm_sql` seed-2, which ships `program.py`, a multi-condition experiment script that runs 8 baseline conditions $\times$ 3 seeds $\times$ multiple datasets at import time before any single number is produced.

`metric_mismatch` (n=2). The paper and the code do not agree on what they are measuring. ARC `llm_sql` seed-3 publishes a headline of 0.0 that the paper itself attributes to "a critical integration failure with the benchmark evaluator framework". The agent reported a known-broken result as its

[1] A small number of auditor false positives were corrected before reporting Table 1. This section reports the flags that survived review. Table 1 reports pass rates over 12 papers per system (EPLB excluded), so its implied failure count is lower than the 23 flags here, which include EPLB papers.

final score. Sakana ASv2 `cloudcast` seed-1 reports per-transfer dollar cost while the evaluator produces a combined score, making the two numbers incomparable. The BFTS metric parser tracked the agent's own metric in place of the canonical score our bridge prints (Appendix J), so the paper reports the unit its run optimized.

Per-system error shape. Table 12 shows that the failure mix is highly system-specific. Sakana ASv2 carries the most flags (8) and the only `cross_stage_cherry_pick` cases, a failure mode unique to multi-stage search pipelines that expose full experiment histories to the writeup phase. Two of its three `value_mismatch` cases and its `metric_mismatch` case arise in re-evaluation or in the adaptation, with no score inflation involved, and its I1 count is left out of the cross-system comparison on that ground. AIR and DS errors are dominated by small-to-medium `value_mismatch`: the numbers exist and are close but do not exactly reproduce. ARC spans the widest range of categories and the largest single discrepancy (106%), and contributes the only `evaluator_error` case. **`ScientistOne`** produces no confirmed I1 errors, consistent with its 12/12 score-verification result in Table 1.

Table 12: I1 flags per system, by class.

System	val_mis.	cherry_pick	score_unavail.	eval_error	metric_mis.	Total
Sakana ASv2	3	4	0	0	1	8
AIR	3	0	1	0	0	4
ARC	5	0	0	1	1	7
DS	3	0	1	0	0	4
`ScientistOne`	0	0	0	0	0	0

ARC. **ARC** matches in 5/12. Its failures have three causes. First, crashed solutions: 5 of 15 import helper modules generated by ARC's multi-file blueprint planner that are absent during standalone re-evaluation, and the evaluator's fallback score then differs from the paper's claim. Second, an evaluator mismatch: ARC's bundled cloudcast evaluator carries a patch the canonical evaluator lacks. Third, unseeded scheduling randomness in txn_scheduling, a 2 to 3% effect.

H.2 I2: Specification Violation Analysis

The I2 audit flagged 11 papers across five systems for specification violations (Table 14). Sakana ASv2 accounts for 10 of 11 flagged papers. The remaining case is an AIR `llm_sql` paper, withdrawn after the blinded human evaluation (§6.2). ARC, DS, and **`ScientistOne`** produce zero I2 violations under majority vote.[2] Most of Sakana's I2 violations trace to the BFTS–ADRS design mismatch (Section J).

Because a single paper can trigger multiple violation categories, we report both the category distribution (Table 13) and the per-system task breakdown (Table 14).

Table 13: I2 violation categories across the 10 flagged Sakana papers. A paper may trigger multiple categories, so paper counts do not sum to 10.

Category	Papers	System(s)
Evaluator import	10	Sakana (10/10)
Evaluator exploitation	7	Sakana (7/10)
Specification exploit	4	Sakana (4/10)
Data leakage	1	Sakana (1/10)

Evaluator import (10/10 Sakana papers). Every flagged Sakana paper imports the canonical evaluator and calls it as an optimization oracle. The mechanism is uniform: the agent adds `from evaluator import evaluate` (or a variant aliased as `_tune_eval`), then loops over hyperparameter configurations, calling `evaluate()` for each, and keeps the best. The import pattern mirrors the canonical evaluation code we injected into the initial code template to enable automated scoring. The agent copies it and builds its own tuning infrastructure on top. The agent could tune

[2] Under union vote (any single judge flags), **`ScientistOne`** has 1, ARC 2, DS 2 and AIR 3. DS seed-1 `llm_sql` was flagged by 2 of 5 judges, below the majority threshold (Case 3, Appendix E).

parameters through BFTS's iteration loop (one setting per iteration, scored by the external evaluator), but the stage 2 goal ("test across multiple parameter settings") encourages an intra-iteration sweep, which requires importing the evaluator directly.

Specification exploit (4 Sakana). Three Sakana `txn_scheduling` papers (seeds 1–3) and one Sakana `llm_sql` paper modify functions outside the `EVOLVE-BLOCK` markers or exploit metric edge cases. The `txn_scheduling` pattern is consistent: the agent modifies `get_random_costs()`, a function explicitly forbidden from modification, to run a parameter sweep and return the best result.

The AIR flag. The audit's one flag among the compared systems (AIR, `llm_sql` seed-1) is a solution that reorders values across columns within each row, which the judges read as inflating the prefix-cache hit metric. Both blinded raters judged it clean, since per-row reordering is the task's own design (§6.2), and Table 1 reports AIR as 0/15. The same pattern appears in DS seed-1 `llm_sql`, below the majority threshold.

Table 14: I2 specification violations per system. For Sakana ASv2, we show the per-task breakdown. Other systems are reported as totals. All counts are the audit's majority votes (3/5 judges) before human review. AIR's one flag was withdrawn after the blinded human evaluation (§6.2).

System	`cloud.`	`eplb`	`llm_sql`	`prism`	`txn_sched.`	**Total**
Sakana ASv2	1/3	0/3	3/3	3/3	3/3	10/15
AIR	0	0	1	0	0	1/15
ARC	0	0	0	0	0	0/15
DS	0	0	0	0	0	0/15
`ScientistOne`	0	0	0	0	0	0/15

Clean runs. Five of 15 Sakana runs produce no I2 violations: all three EPLB seeds and `cloudcast` seeds 2 and 3. EPLB's solution contract is structurally simpler (a pure allocation function with no external scorer dependency), and the two clean `cloudcast` runs show that even on graph-optimization tasks the agent can fulfill BFTS stage goals without importing the evaluator. Under BFTS's stage pressure this is the exception.

H.3 I3: Hallucinated References

The list below shows each distinct hallucinated work, which is the unit Table 1 counts. A work cited by several papers is listed once: ARC's single fabricated entry appears in all three EPLB papers and counts as one. The unit is the work. Work and bibtex key differ in one case: DS's `krajewski2024mixtures` carries two different invented titles in two different papers, so it is one key and two fabricated works.

Hallucinated references

ARC (1)

sutskever2013importance: Sutskever et al. *SGD with Momentum*. ICML, 2013.

AIR (21)

ruth2018castflow: Rüth, Jan et al. *CastFlow: Clean-slate multicast approach using in-advance path computation in software-defined networks*. 2018 IEEE 43rd Conference on Local Computer Networks (LCN), 2018.
dou2022hetmoe: Dou, Shiwei et al. *HetMoE: An Efficient Distributed MoE Training System for Heterogeneous Clusters*. arXiv preprint arXiv:2210.12384, 2022.
li2023lightllm: Li, Zhuohan et al. *LightLLM: A highly optimized LLM inference system with token-level kv cache management*. arXiv preprint arXiv:2309.04414, 2023.

Hallucinated references (continued)

purohit2024prism: Purohit, Archit et al. *Prism: Optimizing multi-model LLM serving on GPU clusters*. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024.
miao2023muxserve: Miao, Xupeng et al. *MuxServe: Multiplexing large language models for high throughput and low latency*. arXiv preprint arXiv:2311.05602, 2023.
cloudcast: Zheng, Q. et al. *CloudCast: Cost-efficient multicast routing in cloud networks*. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019.
idreos2012main: Idreos, Stratos et al. *Main-memory column stores*. Foundations and Trends® in Databases, 2012.
lightllm2023: Li, Zhen et al. *LightLLM: A Lightweight and Highly Efficient Python-based Large Language Model Serving Framework*. arXiv preprint arXiv:2310.01234, 2023.
bhuiyan2024prism: Bhuiyan, M. et al. *Prism: A Flexible and Scalable Multi-LLM Serving System*. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024.
eplb2023: Wang, H. et al. *Expert Parallelism Load Balancing in Mixture-of-Experts Models*. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2023.
hussin2011metaheuristic: Hussin, MS et al. *Metaheuristic algorithms for traveling salesman problem: A review*. Annals of the University of Craiova, Mathematics and Computer Science Series, 2011.
zhang2020job: Zhang, Jun et al. *Job shop scheduling research*. International Journal of Production Research, 2020.
jetstream2014: Rabkin, Ariel et al. *JetStream: Enabling wide-area data streaming*. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), 2014.
slingshot2022: Doe, John et al. *Slingshot: High-performance routing across federated cloud environments*. Proceedings of the ACM SIGCOMM 2022 Conference, 2022.
cloudcast2021: Chen, Yiting et al. *CloudCast: Cost-effective data distribution in multi-cloud deployments*. IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021.
cui2004: Cui, Jun-Hong et al. *QoS multicast routing in dynamic networks*. IEEE Network, 2004.
laoutaris2011netstitcher: Laoutaris, Nikolaos et al. *NetStitcher: Un-tethering bulk storage from the network edge*. Proceedings of the ACM SIGCOMM 2011 conference, 2011.
feng2020traffic: Feng, Xin et al. *Traffic engineering in software-defined wide-area networks: A survey*. IEEE/ACM Transactions on Networking, 2020.
epstein2005online: Epstein, Leah and van Stee, Rob. *Online bin packing with square-root sized items*. Information Processing Letters, 2005.
ba2023modelbox: Ba, Yuhan et al. *ModelBox: A Framework for Multi-Model Multi-Tenant Serving*. 2023 USENIX Annual Technical Conference (USENIX ATC 23), 2023.
zheng2024eplb: Zheng, Lianmin et al. *EPLB: Load Balancing for Expert Parallelism in Large Language Models*. arXiv preprint arXiv:2401.03221, 2024.

DS (42)

choy1991heuristic: Choy, M-S. *Heuristic algorithms for the Steiner tree problem with an application to network routing*. Proceedings of the 1991 ACM SIGCOMM, 1991.
beloglazov2012energy: Beloglazov, Anton and Buyya, Rajkumar. *Energy-efficient routing and resource allocation in multi-cloud*. Journal of Network and Computer Applications, 2012.
grotschel1993steiner: Grotschel, Martin et al. *The Steiner tree packing problem in telecommunications*. 1993.
romero2021automated: Romero, F et al. *Automated algorithm design using large language models*. Advances in Neural Information Processing Systems, 2023.
wu2015spanning: Wu, Chuan et al. *Spanning tree based data transfer in multi-cloud architectures*. 2015.
melian2012cloud: Melian, L et al. *Cloud computing economics: a survey*. 2012.
faragardi2017multi: Faragardi, Hamid Reza et al. *Multi-cloud data distribution with cost optimization*. 2017.
mishra2018cost: Mishra, A et al. *Cost efficient routing in multi-cloud environments*. 2018.
voss1992steiner: Voss, Stefan. *The Steiner tree problem with edge capacities*. 1992.
bubeck2023approaches: Bubeck, S et al. *Approaches to code generation and synthesis*. 2023.

Hallucinated references (continued)

wang2020automated: Wang, X et al. *Automated hyperparameter optimization in cloud computing*. 2020.
smith2019data: Smith, J et al. *Data transfer costs in modern cloud platforms*. 2019.
liu2022network: Liu, Y et al. *Network aware multi-cloud data distribution*. 2022.
jones2023oscillatory: Jones, R et al. *Oscillatory dynamics in automated search landscapes*. 2023.
kim2021puber: Kim, Young Jin et al. *Puber: Efficient expert parallelism for mixture-of-experts*. arXiv preprint arXiv:2111.05454, 2021.
romera2021optimizing: Romera-Paredes, Oscar et al. *Optimizing mixture-of-experts for large-scale distributed training*. arXiv preprint arXiv:2102.04353, 2021.
clune2008how: Clune, Jeff et al. *How evolutionary dynamics affects network topology*. Artificial life, 2008.
llm_agents_2023: Smith, J. and Doe, A. *Autonomous LLM Agents for Code Generation*. Journal of AI Research, 2023.
api_misuse_2024: Wang, L. and Lee, C. *API Misuse in LLM-Generated Code*. Proceedings of ICSE, 2024.
fail_fast_2025: Garcia, M. *Fail-Fast Sandboxing for Coding Agents*. IEEE Transactions on Software Engineering, 2025.
prism2024: Anonymous. *Prism: A Benchmark for Multi-LLM Serving Systems*. arXiv preprint, 2024.
sarca2023: Anonymous. *SARCA: Systems Architecture Research using Coding Agents*. arXiv, 2023.
clockwork2023: Anonymous. *Clockwork: Fast and Predictable Inference for Edge Machine Learning*. arXiv, 2023.
garcia1982fully: Garcia-Molina, Hector. *A fully distributed null-free algorithm for concurrent database updates*. IEEE Transactions on Software Engineering, 1982.
karlsson2020combinatorial: Karlsson, Elias et al. *Combinatorial optimization by graph neural networks*. arXiv preprint arXiv:2010.16012, 2020.
krajewski2024mixtures: Krajewski, Adam et al. *Mixtures of experts: A systematic review*. arXiv preprint arXiv:2401.00000, 2024.
krajewski2024mixtures: Krajewski, Adam et al. *Mixtures of experts past, present and future*. arXiv preprint, 2024. *(same key, different invented title, different paper)*
paliwal2020regal: Paliwal, Aditya et al. *REGAL: A transfer learning based methodology for hardware-software co-design*. MICRO, 2020.
jain1998simulated: Jain, AS and Meeran, S. *A simulated annealing algorithm for job shop scheduling problem*. Mathematical and computer modelling, 1998.
tian2021cloud: Tian, X et al. *Cloud egress cost optimization*. Proceedings of the ACM SIGCOMM 2021 Conference, 2021.
zhao2020understanding: Zhao, Y et al. *Understanding cloud network egress constraints*. USENIX Annual Technical Conference (ATC), 2020.
binnig2021learned: Binnig, Carsten et al. *The case for learned database systems*. arXiv preprint, 2021.
yu2014staccato: Yu, Xiangyao et al. *Staccato: A dependency-aware transaction scheduling system for many-core processors*. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014.
ren2016low: Ren, Kun et al. *Low-overhead deadlock detection in distributed database systems*. Proceedings of the 2016 International Conference on Management of Data, 2016.
pavlo2017make: Pavlo, Andrew and Stonebraker, Michael. *What's make a database system fast?*. ACM SIGMOD Record, 2017.
blanas2010comparison: Blanas, Spyros et al. *A comparison of join algorithms for modern multi-core processors*. Proceedings of the VLDB Endowment, 2010.
bailis2014bolt: Bailis, Peter et al. *Bolt-on conflict-free replicated data types*. ACM SIGMOD Record, 2014.
hardi1992precedence: Hardi, S and Rakow, TC. *Precedence-based transaction scheduling*. Proceedings of the 2nd International Workshop on Research Issues on Data Engineering, 1992.
kim2016fast: Kim, Taesoo et al. *Fast and scalable serializable transactions in multicore in-memory databases*. Proceedings of the 2016 International Conference on Management of Data, 2016.
wu2017transaction: Wu, Yingjun et al. *Transaction scheduling using graph coloring*. Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data, 2017.

Hallucinated references (continued)

lin2015scheduler: Lin, Jianshu et al. *To schedule or not to schedule: When is transaction scheduling worth the overhead?*. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015.
faleiro2017high: Faleiro, Jose M and Abadi, Daniel J. *High performance serializable concurrency control with determinism*. Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data, 2017.

I Every Check by System and Task

Table 1 sums over tasks and seeds. Table 15 opens each sum so a reader can see whether a system's result is carried by one task or holds across all five. **`ScientistOne`**'s reference result holds in every cell: zero fabricated works in all 15 papers, on every task and every seed, against DeepScientist's 42 fabricated works spread over 11 of its 15 papers and AI-Researcher's 21 over 10 papers (5, 7 and 9 by seed). ARC's single fabricated work is one entry from its curated seed library that reaches all three EPLB papers. Two checks carry little information on these systems: I2 flags no paper outside Sakana, and I1 scores 12 of 15 papers per system because EPLB is excluded for everyone.

Table 15: Every check by system and task. I1: papers passing over papers scorable (EPLB excluded). I2: papers flagged out of three seeds. I3: distinct hallucinated works cited in that task, with the number of the three papers that cite at least one in parentheses. A work cited in two tasks appears in both cells, so the per-system total (Table 1) can be smaller than the row sum. The DeepScientist CLOUDCAST seed-3 cell excludes the two fabrications in its recovered bibliography. I4: papers aligned out of three.

Check	System	Prism	Cloudcast	EPLB	LLM-SQL	TXN
I1 ↑	ScientistOne	3/3	3/3	–	3/3	3/3
	AutoResearchClaw	1/3	1/3	–	1/3	2/3
	DeepScientist	3/3	2/3	–	3/3	3/3
	AI-Researcher	2/3	3/3	–	2/3	2/3
	Sakana AI-Scientist-v2	0/3	0/3	–	2/3	2/3
I2 ↓	ScientistOne	0/3	0/3	0/3	0/3	0/3
	AutoResearchClaw	0/3	0/3	0/3	0/3	0/3
	DeepScientist	0/3	0/3	0/3	0/3	0/3
	AI-Researcher	0/3	0/3	0/3	0/3	0/3
	Sakana AI-Scientist-v2	3/3	1/3	0/3	3/3	3/3
I3 ↓	ScientistOne	0 (0/3)	0 (0/3)	0 (0/3)	0 (0/3)	0 (0/3)
	AutoResearchClaw	0 (0/3)	0 (0/3)	1 (3/3)	0 (0/3)	0 (0/3)
	DeepScientist	3 (1/3)	14 (1/3)	6 (3/3)	4 (2/3)	13 (3/3)
	AI-Researcher	8 (3/3)	8 (3/3)	2 (2/3)	1 (1/3)	2 (1/3)
	Sakana AI-Scientist-v2	0 (0/3)	0 (0/3)	0 (0/3)	0 (0/3)	0 (0/3)
I4 ↑	ScientistOne	3/3	2/3	3/3	3/3	3/3
	AutoResearchClaw	2/3	1/3	0/3	0/3	0/3
	DeepScientist	2/3	2/3	1/3	1/3	2/3
	AI-Researcher	2/3	2/3	3/3	3/3	3/3
	Sakana AI-Scientist-v2	1/3	0/3	1/3	2/3	1/3

J Baseline Adaptation Details

All baselines are open-source systems adapted to the ADRS benchmark under identical conditions: Gemini 3.1 Pro backbone, up to 20 solution iterations per task, 2-hour code generation windows, and 3 seeds per task. Runs that failed (API errors, port conflicts between containers, LaTeX errors that produced no PDF) were restarted with fresh state up to three times. No run was restarted to improve its score. Below we detail the per-system adaptations.

Sakana AI-Scientist v2 (Sakana). Sakana v2 required the deepest adaptation. The original codebase assumes an ML-training workflow throughout its 4-stage best-first tree search (BFTS) pipeline: stage goals instruct the agent to "tune learning rates and batch sizes," "introduce datasets from HuggingFace," and "avoid changing the model architecture."

To adapt for ADRS, we rewrote all four stage goals in `agent_manager.py` and 14 prompt locations in `parallel_agent.py`, replacing ML-specific terminology ("training loop," "model architecture," "dataset") with optimization-domain equivalents ("solution search," "algorithm design," "problem instance").

Beyond prompt changes, we created a bridge entry point (`perform_experiments_adrs.py`) that loads ADRS task specifications, injects the evolve-block starter code, and routes evaluation through our task loader in place of Sakana's built-in interpreter. We also added a Gemini 3.1 Pro backend (`backend_gemini.py`), since the original codebase only supports OpenAI-compatible APIs. In total, 16 source files were modified or created, plus 5 task-specific idea files and the conference LaTeX template.

Four post-integration fixes were required, discovered during forensic audit:

1. *Canonical evaluator injection.* BFTS bypasses its own `exec_callback` and executes generated code directly. The evaluation callback we registered was dead code. We instead appended a guarded canonical evaluation wrapper to the initial code template, so every generated `runfile.py` runs the official ADRS evaluator and prints canonical metrics to stdout for the BFTS metric parser. The wrapper evaluates `runfile.py` in the working directory, and solutions that reused it for hyperparameter sweeps do not reproduce under standalone re-evaluation, where that file is absent (§H.1). In one run the parser tracked the agent's own metric in place of the canonical print.
2. *Deterministic best-node selection.* BFTS selects its "best" node via an LLM call at temperature=0.3 that considers "training dynamics" and "plot quality" alongside the raw score. We added a deterministic argmax over all nodes across all 4 stages, producing `best_solver.py` with a fresh canonical re-evaluation.
3. *Paper–code alignment.* The writeup pipeline reads the LLM-selected node, while `best_solver.py` comes from the deterministic argmax, which yielded different algorithms in all 15 runs. The papers evaluated in this study were not regenerated after this fix, so a paper's description may refer to the LLM-selected node while its code is the highest-scoring one.
4. *Plot generation path.* The upstream `load_exp_summaries()` hardcodes `logs/0-run/` as the summary directory, but our bridge writes summaries to the run root. The mismatch caused `aggregate_plots` to receive empty data, producing zero figures across all 15 runs. All 15 Sakana papers are therefore figure-free.

Sakana's specification violation patterns (10/15 papers flagged) are analyzed in §H.2. Three of its eight I1 flags trace to the same adaptation (§H.1). Table 1 reports Sakana's row and leaves it out of the comparison. The dominant cause is a design mismatch between BFTS's multi-stage architecture (which assumes self-contained evaluation) and ADRS's solution contract (which treats the evaluator as off-limits infrastructure).

AutoResearchClaw (ARC). ARC v0.4.0 (commit `39c996b`, April 2026) required 2 source patches: one to add Gemini 3.x to the thinking-model whitelist (2 lines in `llm/client.py`), and one to configure the scaffold directory for ADRS tasks. No changes were made to ARC's search logic, stopping criteria, or evaluation pipeline. We set `max_iterations=20` (6.7× ARC's default of 3) and `max_refine_duration=7200s`. ARC's internal limits (max 2 decision pivots, max 3 repair cycles) are architectural and were not modified. ARC's post-pipeline scoring (PQEP) failed on 4 of 15 runs due to a regex bug in evaluator path substitution. These were resolved by re-running the canonical ADRS evaluator on the submitted code. ARC's `FigureAgent` could not run, because it requires a Docker image not available in our environment.

ARC's blueprint planner designs multi-file project architectures (`config.py`, `models.py`, etc.) before seeing ADRS's single-file constraint. The scaffold partially compensates by renaming the entry point to `program.py`, but the solution still imports helper modules that exist only in the original experiment directory. Re-evaluation in isolation fails for 5 of 15 solutions, the dominant cause of ARC's I1 failures.

Table 16: Discovery configuration for all 15 reported **ScientistOne** runs, read from the per-run configuration dump the orchestrator writes before search begins. Symbols in parentheses are those used in §4.2.

Group	Parameter	Value
Search	Orchestrator	Parallel Explore-Exploit
	Iterations (I)	5
	Parallel branches (B)	5
	Branches retained per iteration (K)	2
	Evaluations per node (E)	4
	Solver early-stopping patience	2 non-improving evaluations
	Concurrent evaluations per node	4
	Branch retries on failure	2
	Seed the first branch with starter code	yes
	Reject solutions the auditor flags	yes
Timeouts	Branch wallclock	1,200 s
	Implementer and ablation agent	900 s each
	Shell command	600 s
Ideator	Conservative share of each idea batch	0.4
	Parallel critics	4
Downstream	Ablations per run	3
	Checkpointer branches	1
Models	Backbone, every agent	`gemini-3.1-pro-preview`
	Ideator, evaluator, auditor, checkpointer	temperature 1.0, thinking level `high`, 65,536 output tokens
	Implementer, ablation agent	provider defaults
	Submission validator	temperature 0.0, 1,024 output tokens
Execution	Solver sandbox	Apptainer container
	Evaluator	host, outside the container
	Node type	8-vCPU c3

DeepScientist (DS). DS v1.5.17 (tag `v1.5.17`, April 2026) on Codex CLI 0.57.0 required prompt-only changes, with no source file patched. Task specifications and the conference template were injected via the write-phase prompt. We set `CODE_TIMEOUT=7200s` and `WRITE_TIMEOUT=5400s`, with up to 3 retries each for code and write phases. DS's write skill instructs the agent to retrieve citations via Semantic Scholar, arXiv, and CrossRef APIs. However, across all 15 write-phase logs, the agent never called any retrieval API or MCP tool, generating all references from model memory. This is a model compliance failure: the tools are available and the agent consistently skips the citation retrieval instructions.

AI-Researcher (AIR). AIR required the most extensive adaptation: 19 source files were patched to interface with ADRS task configurations, the Docker-based sandbox, and the conference paper template. AIR's MetaChain ReAct agent runs inside a Docker container. We configured the same Gemini 3.1 Pro model and iteration budgets as other systems. Of the 15 runs, 6 required re-runs due to Gemini API rate limits or sandbox crashes. Four papers required manual LaTeX fixes (trailing newlines, malformed `filecontents` environments, bibliography corruption).

K System Implementation Details

K.1 Search Configuration

The 15 **ScientistOne** papers in Table 1 come from three campaigns, one per seed, each covering all five ADRS tasks. The orchestrator writes a configuration dump at the start of every run, and the 15 dumps are identical apart from output paths. Table 16 reports every value they record.

Realized search size. Every one of the 15 runs expanded exactly 25 solver nodes, which is $I \times B$ with no early termination of the outer search. Within a node the Implementer holds an evaluation budget of $E{=}4$, shown to it as a countdown and consumed when it saves an evaluation result, which

puts the nominal per-run budget at 100 evaluated solutions. The realized count is a median of 100 (mean 101.2, range 95 to 110), the spread coming from submissions that fail before producing a result and therefore leave the counter untouched.

Prompts. The two Ideator stance prompts, below, are the source of the Conservative and Unconventional labels in Figure 2, and the conservative share of 0.4 in Table 16 sets how each batch is split between them. Braces mark slots filled at call time. The bracketed lines stand in for the prompt's section headings, each of which is one such slot carrying the corresponding block of the PI brief.

```
You are a senior research scientist. Generate **{n}** conservative,
well-grounded research ideas for the task below. Conservative here means:
the idea is a direct, defensible improvement over a SOTA method named in
the brief, targets one of the brief's open problems, and uses techniques
the literature has already validated. Avoid speculative or off-paradigm
ideas in this batch.
  [Problem; SOTA methods and their limitations; open problems; what has
   already been tried and why it fell short; available baselines;
   reference paper pool; previous-round feedback]
Return a JSON array (and nothing else). Each element has the shape:
  {"title": ..., "short_hypothesis": ..., "grounded_in": [...],
   "addresses": ...}
Generate exactly {n} ideas. Each `short_hypothesis` is one or two
sentences stating the testable claim. Do not invent papers --
`grounded_in` entries must appear verbatim in the brief above.

You are a creative research scientist with broad cross-disciplinary
reading. Generate **{n}** unconventional research ideas for the task
below. Unconventional here means: cross-paradigm transfer, reframing the
problem, or attacking an assumption the brief treats as fixed. The idea
must still be runnable on the available baselines and address a real gap
from the brief.
  [Problem; SOTA methods; open problems; what conventional approaches
   have tried; available baselines; reference paper pool for cross-domain
   transfer; How-Might-We seeds; previous-round feedback]
Return a JSON array (and nothing else). Each element:
  {"title": ..., "short_hypothesis": ..., "twist": ...,
   "grounded_in": [...], "addresses": ...}
Generate exactly {n} ideas. Be bold but not unhinged -- every idea must
remain empirically testable against the brief's baselines.
```

The Implementer's standing instructions govern the inner loop. We reproduce them below with the tool-mechanics detail (argument shapes, return-field names, file layout) trimmed, leaving the job-queue protocol, the logging obligation and the stopping rule intact.

```
You are an expert AI research assistant. Your goal is to explore,
evaluate, and iteratively refine the best solution code based on a
provided idea and task description.
- Solution Generation: Generate a solution based on the selected Idea
  and task descriptions. Use the proposed idea as your primary guide,
  but adapt details if evidence suggests a better path.
- Use the Job Queue: Submit your solution code using the `submit_job`
  tool. Submit multiple jobs concurrently -- keep submitting as long as
  the number of pending or running jobs is below the maximum concurrent
  job limit. Prepare and submit diverse solution variants in parallel to
  maximize exploration.
- Check Status & Logs: Use `check_job_status(top_k=...)` periodically.
  Pay close attention to the evaluation metrics, standard output logs,
  and any error traces in the results. Whenever `available_slots > 0`,
  you MUST immediately prepare and submit new jobs to fill those slots.
- Iterative Improvement: If a job fails or has poor metrics, deeply
  analyze the logs, identify the root cause, and formulate a targeted
  fix before submitting the next iteration.
- Lessons Learned & Reflection: Actively summarize lessons learned from
  previous solutions, both successes and failures, and use them to guide
```

```
  the next iteration.
- Experiment Logging: After every iteration, append a concise entry to
  `log.txt`. The log is the only durable record of what was tried and
  will be the primary input to a downstream report-writing agent. Each
  entry should include: iteration number, the hypothesis or change being
  tested (and how it differs from prior attempts), the relevant
  `solution_{id}` folders, the resulting metrics, what worked or failed,
  and the takeaway that informs the next iteration. Always append.
- Evaluation Budget: `check_job_status` returns an `eval_budget` field
  showing how many evaluations you have used and how many remain. Plan
  strategically -- explore broadly early, then refine. When the budget
  is exhausted, `submit_job` will reject further submissions.
- Early Stopping: `check_job_status` returns an `early_stopping` field
  reporting the current best metric, how many consecutive iterations
  have passed without improvement, and how many remain. When it
  indicates that early stopping has been TRIGGERED, you MUST
  immediately stop submitting new jobs and finalize your work.
- Finalize: The best solution code and evaluation results are saved
  automatically. Make sure `log.txt` contains a final entry covering the
  last iteration. A separate downstream stage produces the report -- do
  not write any report yourself.
```

The remaining prompts are stage-specific and described where the stage is: the Paper Writer's five stages in Appendix K.5, the Claim Verifier's dispatch rules in Appendix K.6, and the four audit judges in Table 7.

K.2 Problem Investigator

PI is a five-stage pipeline (plus two auxiliary stages), where each stage communicates via file-backed artifacts on disk.

Stage 1: Citation Graph. Starting from 2–4 seed papers, PI traverses the Semantic Scholar API (references and citations) up to 2 hops deep, producing a citation graph of approximately 2,000–5,000 candidate papers.

Stage 2: Literature Filter. An LLM scores each paper in the graph on two axes, methodology relevance and problem alignment (1–5 each), and classifies papers into tiers: Core (both $\geq$4), Adjacent (one $\geq$4, other $\geq$3), Spark, or Noise. The resulting elite pool contains approximately 500 papers. A topic-relevance gate aborts the pipeline if fewer than 5 Core+Adjacent papers appear, preventing downstream drift from weak seeds.

Stage 3: Multi-Round Investigation. A Principal Investigator agent orchestrates specialist sub-agents across 3 rounds. Each round consists of candidate selection from the elite pool, parallel PDF reading and structured note extraction (5 Researcher agents), and synthesis of findings into thematic research direction dossiers (SubdomainWriter agent). Candidate selection is performed by the *Librarian*, the sub-agent that queries the scholarly database, maintains the elite pool of retrieved papers, and decides which of them are read in full in the next round. An IslandConsolidator agent merges redundant directions and retires low-quality ones after each round. The target is approximately 100 paper notes organized into 5–15 research directions.

Stage 4: Evaluation Protocol Audit and Targeted Literature Refresh. Per-direction audit reports are generated and scored on a checklist rubric across multiple rounds until the direction passes. A focused mini citation crawl on the winning direction produces 20–30 additional paper notes, filling gaps identified during the audit.

Stage 5: Experiment Brief Synthesis. Directions are scored by seed relevance, then a section-by-section writer produces the final Experiment Brief through a multi-round critic loop (up to 5 rounds with section-level revision). The brief contains three sections: (1) a research landscape with technique taxonomy and best-known results, (2) a concrete experiment plan with baselines, metrics, and ablation design, and (3) a literature context with 25–40 references traceable to paper notes extracted from the source PDFs.

K.3 IMPLEMENTER

The Implementer consists of two agents. The *Solution Development Agent* receives an idea and task-specific instructions, operates in a sandboxed environment with tools for file I/O, command-line execution, solution management, and knowledge base retrieval, and follows an iterative refinement loop (executing experiments, debugging errors, and optimizing validation metrics) while maintaining an experimental log. The *Report Writing Agent* parses experimental artifacts to generate a technical report summarizing methodology and outcomes.

K.4 ABLATION

Following PEE, the best-run selector filters out solutions flagged for specification violations and selects the highest-scoring remaining solution for further validation. An ablation agent identifies the solution's core components and proposes controlled experiments to isolate their contributions. These ablated versions are implemented and re-evaluated to quantify the performance impact of specific architectural or algorithmic choices.

K.5 PAPER WRITER

The Paper Writer is a five-stage pipeline that converts raw materials into a verified LaTeX draft. The first four stages operate on a *research representation*, a structured markdown narrative with inline evidence annotations, before any LaTeX is generated, so that provenance is fixed before the text is written (§4.3).

CONCEIVE. A single LLM call reads all assembled raw materials (PI brief, experimental log, evaluator scores, solution code, seed-paper abstracts) and produces the initial research representation. This document captures the narrative (problem, gap, approach, result, limitation), with every factual claim carrying an inline evidence tag binding it to a specific workspace artifact (log line, score file, citation key, or ablation entry). CONCEIVE fixes the narrative structure. GROUND validates the evidence chains.

GROUND. Deterministic checks validate each evidence annotation against the raw materials: the reported score must match the best-run score from discovery, baselines are labeled VERIFIED (traceable to a PI brief entry) or ESTIMATED (unattributed "leaderboard" references are flagged), every referenced artifact must exist, all expected sections are present, and hyperbole counts and known score mismatches are recorded. Each claim receives a SUPPORTED, PARTIAL, or UNSUPPORTED label, and the overall grounding ratio (supported claims / total claims) is computed.

CRITIC. One LLM call audits story-level coherence: gap–approach alignment, internal contradictions, overclaims relative to evidence strength, missing comparisons, baseline fairness, and whether limitations are stated. The critic returns PASS or a list of MAJOR/MINOR issues.

RESOLVE. A single LLM call rewrites the representation against the GROUND flags and CRITIC issues jointly: unsupported claims are dropped or softened, contradictions are resolved using the verified source, overclaims are calibrated, and missing sections are filled. The loop over GROUND, CRITIC and RESOLVE iterates for up to two rounds, terminating on convergence (zero flags) or plateau (the flag count stops decreasing). A grounding ratio that remains below a configured threshold aborts the run, and no draft is produced.

COMPOSE. The grounded representation is handed to per-section writers that emit LaTeX one section at a time, with each numerical or citation-bearing sentence committing to its evidence source at writing time. The composed draft then passes through the Claim Verifier (Appendix K.6). A refinement pass consumes the verifier's findings, rewriting flagged sentences to match their evidence sources, removing unsupported claims, and stripping all inline evidence annotations from the final LaTeX. The draft is final only when no blocking violation remains.

K.6 CLAIM VERIFIER

The Claim Verifier applies one verification rule per claim type (§3). The writer commits each claim's evidence via annotation tags (a line in the experimental log, a citation key, an internal ablation, a

baseline from the PI brief, or an explicit "unsourced" marker), and the verifier maps that evidence onto the claim type's verification rule:

- **Numerical claims** are checked by numeric tolerance against the cited evidence (log line, ablation entry, or PI baseline), with a ± 3-line window on log lines and unit-aware normalizations for percent-versus-fraction and millisecond-versus-second mismatches.
- **Citation claims** are checked by resolving the cite key against the bibliography and then asking a one-shot LLM judge (JSON mode) whether the cited work's abstract supports the specific assertion. This check is internal to the writer. The audit of §5 does not read it and verifies reference existence only, since the baseline papers carry no claim-to-citation record.
- **Methodological claims** are checked by substantive textual overlap against the cited region of the experimental log.

Claims tagged "unsourced" or carrying malformed annotations are dropped automatically. A break code is recorded for each for downstream reporting.

K.7 Cost and Runtime

We instrumented every model call in the pipeline with the module, agent role, and iteration that issued it, and price each call at the rate of the model that served it, which matters because prompts above 200K tokens bill at double the base rate. All dollar figures are Vertex AI list prices at the Standard tier on the global endpoint, with no committed-use discount. Cache *storage* is excluded, because we record cache reads and have no record of cache lifetime. Preview-model prices move, so these are the costs of the reported runs at the time they ran, and a current quote may differ.

Table 17: What one **`ScientistOne`** paper costs, by stage. Vertex AI list prices at the time of the runs, no committed-use discount. Details in Appendix K.7.

Stage	Tokens	Model calls	Wallclock	Cost (USD)
Literature review, per task	9.3M	–	8.9 h	12.61
Solution discovery, per run	362.9M	4,600	117 min	261
Paper writing, per paper	240k	16	–	0.59
of which claim verification	2.1k	–	–	$<$0.01
CoE Integrity Audit, per paper	182k	24	4 min	0.49

Discovery dominates, and the solution search is nearly all of it. A complete discovery run averages 362.9M tokens in about 4,600 API calls, with under 2% spread across LLM_SQL, PRISM and CLOUDCAST. The Implementer accounts for 99.94% of those tokens. Ideation, the specification-violation auditor and the checkpointer together come to under 0.1%. Prompt caching serves 76.1% of input tokens, so billable input is roughly a quarter of the raw count, and caching accounts for a factor of three in the totals below: the same run would bill $782 without it.

The literature stage is a per-task cost. The Problem Investigator consumes 9.3M tokens over 8.9 hours, the longest wallclock of any module and 2.6% of a single discovery run's tokens. It runs once per task and its brief is then reused unchanged. For CLOUDCAST, all 40 runs in our archive consume an identical brief across seeds, configurations, and three months of experiments. Amortized over those runs it is 232k tokens, 0.06% of what one discovery run spends, and it is off the per-run critical path.

Paper writing, and the verification inside it. Across 280 writer invocations the mean is 240k tokens per paper (216k input, 24k output) in 16 API calls, three orders of magnitude below discovery. Claim verification, verification-driven repair and source checking together account for about 0.9% of that, roughly 2,100 tokens per paper.

Totals. A discovery run costs about $261, the literature stage $12.61 per task, and paper writing $0.59 per paper, with the literature stage spread over a task's three seeds, so roughly $266 produces one paper and about $3,990 produced all 15 **`ScientistOne`** papers in Table 1. These are production costs for the reported results. A research budget would add the baselines, the ablations, and every superseded run. Median discovery wallclock is 117 minutes per run (mean 128, n=40), from 104 minutes on EPLB to 176 on LLM_SQL, on 8 vCPU c3 nodes.

The audit. Re-running CoE Integrity Audit over all 15 papers at the vote depth used in the paper consumes 182k tokens per paper in 24 model calls, which prices at $0.49. That is 0.05% of the tokens and 0.19% of the dollars that produced the paper it audits. Throughput is 238 seconds per paper (15 papers in 60 minutes on one 8-vCPU node, three in flight). Method-code alignment and specification-violation checking account for most of that cost, claim extraction for about a tenth of it, and reference verification costs almost nothing in model terms because API lookups dominate it.