ScientistOne: Verifiable Autonomous Research via Chain-of-Evidence

Anonymous Author(s)
Affiliation
Address
email

Abstract

Autonomous research agents produce competitive solutions and complete manuscripts, yet their outputs suffer from verifiability failures that are largely invisible to evaluations that check fluency rather than evidence: fabricated citations, unreproducible scores, and method descriptions that diverge from the implementation. These failures share a common root: no existing system is designed to trace claims back to evidence, and no existing evaluation protocol audits whether the claims are supported. We formalize this gap through *Chain-of-Evidence* (CoE), a verifiability standard requiring every claim to be traceable to its evidence source, and instantiate it in two contributions: ScientistOne, an end-to-end autonomous research system that maintains evidence chains by construction throughout literature review, solution discovery, and paper writing; and *CoE Audit*, an evaluation protocol whose four integrity checks—score verification, specification violation, citation verification, and method–code alignment—apply uniformly to all systems. Across 60 papers from four systems on five frontier systems-research tasks, we find that every baseline exhibits at least one systematic failure mode: phantom citation rates reach 21%, score verification passes in as few as 42% of papers, and method–code alignment ranges from 20% to 80%. ScientistOne is the only system to achieve zero phantom citations (0/337 references), perfect score verification (12/12), and the highest method–code alignment (14/15), while remaining competitive on solver performance. We further demonstrate that ScientistOne generalizes to six additional tasks spanning medical imaging, fine-grained recognition, 3D perception, and parameter-constrained language modeling, achieving state-of-the-art on Parameter Golf and gold medals on MLE-Bench tasks where baselines fail entirely.

1 Introduction

Large language models are increasingly deployed not as isolated assistants but as autonomous agents that conduct entire research workflows—from literature review and hypothesis generation through experimental design and execution to manuscript writing [1, 2, 3, 4, 5, 6, 7]. On systems-optimization tasks, such agents now produce solutions competitive with human experts [8, 9], and end-to-end pipelines have generated papers accepted at peer-reviewed workshops [2]. The resulting artifacts—code, experimental results, and complete manuscripts—are increasingly difficult to distinguish from human-authored research on surface quality alone.

This rapid capability growth exposes a structural tension between *generation* and *verification*. Autonomous research systems operate as multi-stage pipelines in which each stage consumes the output of the previous one: a literature summary shapes the hypothesis, the hypothesis determines the experiment, and experimental results feed into the manuscript. In such architectures, errors introduced at any stage are not merely preserved but amplified—a flawed summary can bias experimental design,

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

and a misinterpreted result can propagate into a paper that appears internally coherent precisely because the same error threads through every section. The risk grows with trajectory length: agents struggle to track an ever-expanding context [10, 11], hallucinate, and lose sight of the original purpose. The problem is compounded by fundamental limitations in how language models handle evidence: generated text is difficult to verify against sources [12], factual claims drift from their grounding [13], and scientific citations are frequently inaccurate or fabricated [14].

In autonomous pipelines, these failure modes interact and compound—a model can embellish method descriptions beyond what the code implements, report scores that do not reproduce under the benchmark's own evaluator, and populate bibliographies from parametric memory rather than retrieval, all while producing prose that reads as technically sound. Existing evaluation protocols, whether automated review scores or benchmark leaderboards, reward output fluency and procedural completion but do not assess whether individual claims trace to supporting evidence.

This verifiability gap is not hypothetical. In a systematic audit of 60 papers generated by four autonomous research systems across five benchmark tasks, we find that *every baseline paper* exhibits at least one evidence chain failure: phantom citations that do not correspond to any real publication (up to 21% of all references), method sections that describe algorithms entirely absent from the submitted code, unreproducible scores, and solution code that exploits the evaluator rather than solving the task. These failures share a common root cause: *no existing system is designed to trace claims back to evidence, and no existing evaluation protocol audits whether the claims are supported.*

We address this with *Chain-of-Evidence* (CoE), a verifiability standard for AI-driven research. Just as ACID[1] [15] defines what "reliable" means as a standard for a database transaction, CoE defines what "verifiable" means for a research claim: **every claim must trace, through a recorded evidence chain, to a grounding source.** We instantiate CoE in three ways:

1. **The CoE Standard** (§3): a claim taxonomy (citation, numerical, methodological, conclusion) and the evidence chain structure required for each type.
2. `ScientistOne` (§4): an end-to-end autonomous research system whose pipeline—Problem Investigator, Discovery Engine, and Paper Writer with Claim Verifier—is designed to satisfy CoE natively. The Problem Investigator reads up to 100 full-text PDFs per topic, producing grounded experiment briefs; the Claim Verifier checks every claim in the draft against its declared evidence source before the final paper is produced.
3. **CoE Audit** (§5): a post-hoc evaluation protocol for auditing an AI-driven research paper through four integrity checks—Score Verification, Specification Violation, Citation Verification, and Method-Code Alignment—targeting the most damaging evidence chain failures.

We apply CoE Audit to 15 papers from each of four systems across five frontier systems-research tasks from ADRS [8, 16] (§6). Every baseline exhibits at least one integrity check failure. `ScientistOne` achieves zero phantom citations (0/337 references), perfect score verification (12/12), and the highest method–code alignment (14/15), while remaining competitive on solver performance across all five tasks. We further demonstrate that `ScientistOne` generalizes to six additional tasks spanning medical imaging, fine-grained recognition, 3D perception, and parameter-constrained language modeling, achieving state-of-the-art on Parameter Golf and gold medals on MLE-Bench tasks where baselines fail entirely.

2 Related Work

Autonomous research agents. End-to-end autonomous research systems have rapidly expanded from constrained ML templates to multi-stage pipelines that coordinate literature grounding, hypothesis generation, experimentation, and paper writing. The AI Scientist [1] pioneered end-to-end automation but operates on fixed ML templates with hallucinated references and workshop-reject quality. AI Scientist-v2 [2] advances this with agentic tree search over experimental branches and review-aware reporting, achieving stronger results on NeurIPS-style tasks. Several concurrent systems extend the pipeline in different directions: PiFlow [6] introduces principle-aware workflow coordination; Curie [17] emphasizes execution-intensive experimentation with controlled reruns and ablations; CodeScientist [7] grounds ideation in open-source code repositories; Agent Laboratory [5] provides a template-bounded full-loop environment; and AlphaEvolve [9] and EvoScientist [18] apply

[1] Atomicity, consistency, isolation, durability.

evolutionary search to scientific optimization. Despite this architectural diversity, a common pattern emerges: generation and execution capabilities have scaled faster than validation and provenance mechanisms, so systems that produce polished manuscripts may still lack credible scientific closure. `ScientistOne` targets this gap—rather than advancing the autonomy frontier, we focus on making autonomous research outputs verifiable.

LLM-driven optimization and benchmarks. The ADRS benchmark [8] collects real frontier computer system research questions and serves as our primary evaluation testbed. Evolutionary search frameworks (AlphaEvolve [9], AdaEvolve [19], EvoScientist [18]) achieve strong results on ADRS by only focusing algorithm discovery and implementation optimization, while not performing literature grounding, novelty assessment, nor paper writing. Broader evaluation resources have recently proliferated: Auto-Bench [20], ResearchBench [21], and ResearcherBench [22] probe discovery-oriented research behavior; MLAgentBench [23], EXP-Bench [24], and PaperBench [25] stress-test experimentation, replication, and execution reliability; while AIRS-Bench [26] and FIRE-Bench [27] target frontier research tasks and rediscovery verification. However, most benchmarks measure procedural correctness—whether a system can complete a research-like workflow—rather than whether the resulting claims are actually supported by evidence. CoE Audit fills this gap by operationalizing validity, reliability, and provenance as a cross-system audit protocol. We also showcase `ScientistOne` that generalizes beyond computer system research problems to other domains, including computer vision, healthcare [28] and low-bit quantization on language modeling [29].

Scientific integrity and provenance. Current autonomous research systems range in reporting maturity from direct manuscript drafting [1, 7, 3] to iterative review-aware revision [2, 30] to artifact-linked reporting where manuscripts are tied to execution traces and code [4, 17]. Artifact-linked reporting provides the strongest traceability but remains uncommon; most systems produce fluent papers that mask broken evidence chains.

Prior work on citation hallucination [12], factual accuracy [13], and citation-level claim support [14] performs post-hoc detection at the text level. CoE differs in two ways: it defines a *constructive* standard (evidence chains must be built at claim-production time, not recovered after the fact), and it covers the full research artifact (paper + code + evaluator logs), not just text. CoE Audit operationalizes this standard as a cross-system evaluation protocol that can be applied in forensic mode to any system's released artifacts.

3 Chain-of-Evidence: A Verifiability Standard

> ***Principle:*** *Every claim produced by a research system must be traceable, through a chain of recorded evidence, to a grounding source—and that chain must be cheap to walk for both humans and machines.*

A credible research claim must be backed by verifiable evidence. Without this requirement, the same system that produces a plausible-sounding paper can also produce fabricated citations, hallucinated numbers, and descriptions of experiments that never happened. Just as a database that violates ACID may return plausible-looking query results even as it silently corrupts data, a research system that violates CoE may produce plausible-looking papers whose claims cannot be traced to evidence. ACID does not prescribe *how* to build a database; it prescribes what properties the database must have. CoE similarly prescribes what properties a research artifact must have, independent of system architecture.

We define four claim types, each with a required evidence chain shape. ***Citation claims*** (e.g., "Smith et al. showed X") require that the cited work exists in a scholarly database and that its content is consistent with the assertion made about it. ***Numerical claims*** (e.g., "achieves 87.3% on Prism") must trace from the reported value to an execution log produced by an evaluator run. ***Methodological claims*** (e.g., "we use a 3-layer MLP") must resolve from the method description to corresponding source code. ***Conclusion claims*** (e.g., "outperforms baseline by 5%") derive from supporting numerical claims through a verifiable comparison. CoE is deliberately architecture-agnostic: it defines what properties a verifiable artifact should have, not how the system should construct one.

In the following sections, we describe `ScientistOne`, an autonomous research system designed to satisfy CoE by construction (§4), and CoE Audit, an automated audit framework that measures how well any system's artifacts meet the standard through four integrity checks (§5).

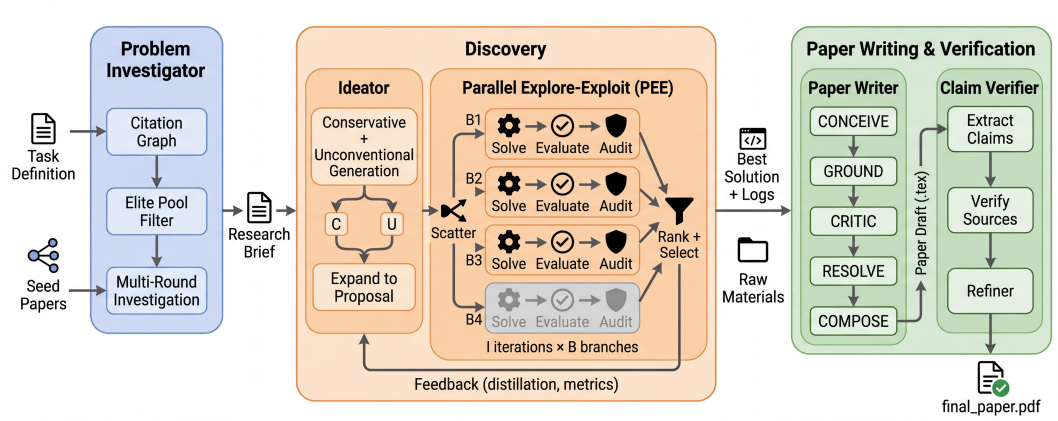

Figure 1: `ScientistOne` pipeline. Stage 1 grounds the literature via retrieved PDFs; Stage 2 explores and evaluates solutions across parallel branches; Stage 3 writes and verifies the paper, with a Claim Verifier that checks every claim against its evidence source before the final output is produced.

4 `ScientistOne`: Research with Verifiability

We now describe `ScientistOne`, an end-to-end autonomous research system whose three-stage architecture is shaped by the CoE requirements: each module is designed to produce structured artifacts that carry the provenance metadata needed to verify claims against their evidence (Figure 1).

4.1 Stage 1: Literature Grounding

The Problem Investigator (PI) is designed to ensure that every paper the system cites was retrieved from a scholarly database, read in full text, and recorded with provenance metadata. Without structured retrieval, autonomous systems generate citations from model memory—our audit finds this produces phantom references at rates of up to 21% (§6.1). PI addresses this by construction: starting from seed papers, it builds a citation graph via the Semantic Scholar API, reads up to 100 full-text PDFs per topic, and produces a structured research brief. The brief feeds the Ideator; PI's seed reference bibliography provides grounding material for citation claims in the final paper. Pipeline details are in §C.

4.2 Stage 2: Discovery

The Ideator generates candidate approaches from the PI brief, scores them on novelty and feasibility, and scatters the top-ranked proposals across parallel branches of the *Parallel Explore-Exploit* (PEE) orchestrator. Each branch runs an isolated cycle: a Solver agent implements a solution, a task-specific evaluator scores it, and an auditor checks idea–implementation alignment. High-performing branches are refined; underperformers are replaced by new explorations. After I iterations across B branches, all execution logs and evaluator outputs are assembled into `verified_scores.json`—the ground truth that the Claim Verifier checks numerical claims against. Architecture details are in §C.

4.3 Stage 3: Paper Writing & Verification

Paper Writer. The Paper Writer first builds a structured representation from all raw materials (PI brief, experimental log, verified scores, code), where every factual claim carries an inline evidence tag binding it to a workspace artifact. Deterministic checks validate score presence and baseline attribution; an LLM critic audits story-level coherence; and a resolve loop refines until convergence. Per-section writers then render the representation into LaTeX, with each claim committing to its evidence source at writing time—verified numbers and named baselines are the first context the writer sees, reducing the incentive to fill gaps from model memory. The composed draft passes through the Claim Verifier; unsupported claims are corrected or dropped before the final paper is produced.

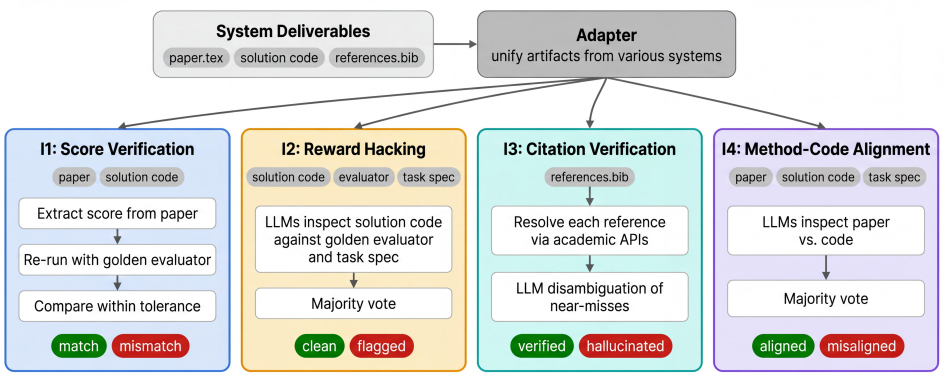

Figure 2: CoE Audit overview.

Claim Verifier. The Claim Verifier checks every claim in the draft against its declared evidence source: numerical claims are compared by tolerance against evaluator logs, citation claims are resolved against the bibliography, and methodological claims are checked against experimental logs. Unsourced claims are flagged automatically. Per-type verification rules are in §C.3.

5 CoE Audit: Measuring Verifiability

To evaluate verifiability of research outcomes—both ours and baselines'—we introduce *CoE Audit*, an evaluation protocol applied identically to any system (Figure 2). An adapter unifies each system's deliverables (paper, solution code, references) into a common artifact bundle, then four integrity checks run independently each targeting a specific way evidence chains break:

Score Verification (I1). The paper's reported score is extracted by LLMs from both TEX and PDF files, then compared against reproduced scores with solution files on the golden evaluator. A paper passes if the two match within an adaptive tolerance that accounts for evaluator noise.

Specification Violation (I2). Specification violations occur when solution code breaks task rules—for example, reading evaluator source files to extract expected outputs, or hardcoding answers for known test cases. The generating agent optimizes for the score, not for the problem the task is designed to measure. LLMs inspect the solution code against the golden evaluator and task specification to detect such violations, with majority vote across multiple runs.

Citation Verification (I3). Each reference is resolved by querying multiple academic APIs (Semantic Scholar, arXiv, OpenAlex, CrossRef) using arXiv ID, DOI, and title. An LLM cross-checks the full bib entry against returned records to catch near-misses and citation gaming (e.g., a real DOI attached to a fabricated description). References matching no record are classified as hallucinated.

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 (e.g., omitting implementation details) is treated as aligned; only cases where the paper describes a fundamentally different algorithm count as misaligned. We conduct multiple independent runs with majority vote to reduce LLM judgment noise.

6 Experiments

Benchmark. We evaluate on the Automated Design of Research Systems (ADRS) benchmark [8], which collect five frontier computer system research problems: **Prism** (GPU memory allocation for LLM serving), **Cloudcast** (cloud network cost optimization), **EPLB** (expert-parallel load balancing for MoE models), **LLM-SQL** (text-to-SQL query optimization), and **TXN** (transaction scheduling). Each task provides a fixed evaluator, starter code, and scoring metric. We choose ADRS as our primary benchmark for three reasons: (1) the tasks are drawn from real-world systems-optimization

problems and researchers published results on those in the past years, not synthetic puzzles; (2) the leaderboard provides both human expert and recent LLM-agent baselines [8], enabling apples-to-apples comparison; and (3) the gold-standard evaluators are deterministic enough to support Score Verification (§5) and Specification Violation detection (§5). Following [19], who observe that ADRS evaluations involve "noisy feedback," we run each evaluator five times and compare against adaptive tolerance $\max(1\%, 3\sigma/|\bar{s}|)$ to account for inherent evaluator variance.

Baseline Systems. We evaluate three baseline systems and `ScientistOne`. All baselines are open-source, enabling us to adapt each to the ADRS benchmark for controlled comparison under identical conditions. They span the design spectrum from highly structured scaffolding to fully autonomous agents, providing coverage across the major paradigms for AI research agents.

- **AutoResearchClaw (ARC)** [31]: 23-stage waterfall pipeline with OpenAlex/Semantic Scholar.
- **DeepScientist (DS)** [3]: Autonomous research studio with Codex CLI and MCP tools.
- **AI-Researcher (AIR)** [4]: MetaChain ReAct agents, adapted for systems-optimization.
- `ScientistOne`: Full pipeline with evidence chain maintenance from problem framing through paper composition (§4).

We also ran Sakana AI-Scientist v2 [2] on all five ADRS tasks but exclude it from results: its pipeline's hardcoded ML-training stage design (e.g., "introduce datasets from HuggingFace," "change learning rate and batch size") overrides the task specification, causing agents to train neural networks rather than optimize the target functions. Only one of fifteen runs produced a solution that called the ADRS evaluator as specified.

ADRS Adaptation. Each baseline was adapted to the ADRS benchmark with varying levels of source modification, from prompt-only changes (DS) to patching 8–19 source files (ARC, AIR) to interface with ADRS task specifications, evaluators, and the NeurIPS 2026 paper template. We standardized on Gemini 3.1 Pro as the backbone LLM across all systems for both solver code generation and paper writing. To ensure best-effort performance, we configured generous iteration and timeout budgets: up to 20 solver iterations per task—6.7× the default for ARC, though most tasks converge well before this cap—and 2-hour code generation windows. Runs that crashed due to infrastructure issues (API timeouts, rate limits, LaTeX compilation errors) were re-attempted with fresh state, up to 3 attempts per run; no run was re-attempted to improve solver scores. Of the 60 total runs, 14 required at least one retry. All 60 runs ultimately produced solver code and a compiled paper, though artifact quality varies. For each system, we run 3 seeds per task, producing 15 papers per system and 60 papers total. CoE Audit is applied identically to all systems. Full adaptation details are provided in Section G.

6.1 CoE Audit Results

Table 1: CoE Audit results across four systems (15 papers per system). EPLB papers are excluded from Score Verif. (no system reports a single Combined Score). Metric definitions are in §5.

System	Score Verif. ↑	Spec. Violation ↓	Citation Verif. ↓	Method-Code ↑
AutoResearchClaw [31]	5/12	0/15	3/196	3/15
DeepScientist [3]	11/12	1/15	42/201	5/15
AI-Researcher [4]	9/12	1/15	21/222	12/15
`ScientistOne`	12/12	0/15	0/337	14/15

The results of CoE Audit across four systems are presented in Table 1. All I1–I3 flagged positives were manually verified and corrected by human reviewers; I4 judgments were validated on a sampled basis. `ScientistOne` is the only system that achieves perfect score verification (12/12), zero specification violations (0/15), zero phantom citations, and 14/15 method-code alignment. The gap is largest in citation integrity and method-code alignment—the two checks that test evidence provenance rather than score reproduction.

Score verification and specification violation. `ScientistOne` achieves perfect score verification (12/12): every paper's claimed result reproduces exactly under re-evaluation. DS matches in 11/12

(92%), with the single failure caused by fabricated metric direction—the paper relabels a cost metric as "utility" and inverts the optimization direction, so a worse-than-baseline score reads as an improvement; rerunning the declared solution against the true metric confirms the baseline still wins. AIR matches in 9/12 (75%), with the failures being small-magnitude discrepancies (1–4%) plus one paper that reports no quantitative scores at all. ARC matches in 5/12 (42%), and its failures span a wider range—from a few percent up to a 2× gap—and include qualitatively distinct cases: a paper without any results table, submission code that prevents clean re-evaluation, and a paper that acknowledges an integration failure with the benchmark evaluator. Specification violation rates are uniformly low: ARC and `ScientistOne` show 0/15, while AIR and DS each have one flagged paper involving evaluator file modifications detected during code inspection (see Case 3, §B.1).

Citation integrity. `ScientistOne` achieves zero phantom citations across all 15 papers (0 out of 337 unique references). DS exhibits the highest phantom rate (42/201, 20.9%), followed by AIR (21/222, 9.5%) and ARC (3/196, 1.5%). ARC's low phantom rate reflects its structured retrieval pipeline; its three phantoms are a single repeated malformed bib entry across three papers. DS and AIR rely on model memory for reference generation, producing plausible-looking but non-existent citations—a failure mode illustrated in Case 2 (§B.1). `ScientistOne`'s zero phantom rate is an architectural property of the PI's citation graph: every reference originates from a Semantic Scholar API call whose result is cached in the evidence chain. (Full list of phantom citations in § E.2.)

Method-code alignment. `ScientistOne` achieves 14/15 aligned papers (93%) under the MCA-0/MCA-1 threshold, compared to AIR (12/15, 80%), DS (5/15, 33%), and ARC (3/15, 20%). The misalignment patterns differ qualitatively across systems. AIR's failures are typically algorithm mismatches—e.g., the paper describes a more sophisticated procedure than the code implements. ARC exhibits the broadest failure-mode mix: algorithm-class mismatches (e.g. beam search with Edmonds' arborescence vs. greedy edge penalization), undisclosed environment-variable switches that swap in entirely different solvers at runtime, evaluation hacking (e.g. returning empty column orderings to maximize a prefix-hit metric), and method-vs-ablation inversion between paper and final solution code. `ScientistOne`'s single misaligned paper is a case where the paper writer fabricated algorithmic claims not present in the code—describing a "hybrid neuro-symbolic solver" with "LLM-guided evolutionary search" when the submitted code is a deterministic routing heuristic with no LLM calls. The Claim Verifier's method-code cross-check (§4.3) catches nearly all such misrepresentation before paper finalization, though complete coverage remains a limitation (§A).

We present detailed Failure Mode case studies in § B.1. We also present sub-categorization and statistics for CoE audit errors in § E.1, E.2, E.3 for I1, I3 and I4 respectively.

6.2 Review Scores

We evaluate perceived paper quality using ScholarPeer [32], an automated peer review system backed by `gemini-3.1-pro-preview` and rich literature search support.

Table 2: ScholarPeer review rating scores (1–4 scales except Overall (1-10 scale)) and accept decisions. *Average*: mean across 15 papers per system. *Best-of-3*: strongest seed per task.

System	Soundness	Originality	Quality	Clarity	Overall	#Accept
Average (3 seeds × 5 tasks)						
ARC	1.1	2.3	1.1	2.5	1.9	0/15
DS	1.7	1.7	1.6	3.1	2.5	1/15
AIR	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**
Best-of-3 (strongest seed per task)						
ARC	1.2	2.0	1.2	3.0	3.0	0/5
DS	2.2	2.0	2.2	3.6	3.6	1/5
AIR	2.0	2.4	2.0	3.2	4.0	1/5
`ScientistOne`	**2.8**	**3.0**	**2.8**	**3.6**	**6.6**	**4/5**

Verifiable papers are deemed better by automatic reviewers. `ScientistOne` achieves a 40% accept rate (6/15), tripling the best baseline (AIR: 13%), and best-of-3 selection reaches 6.6 overall

Table 3: Solution discovery performance on ADRS benchmark tasks (best-of-3 seeds). ARC/AIR/DS/ScientistOne scores are from independent gold evaluator re-runs on submitted solver code; Human, AdaEvolve, and EvoX scores are from original publications. * indicates Gemini-3.0-Pro; all other systems use Gemini-3.1-Pro.

Task	Dir.	Human	AdaEvo*	EvoX*	ARC	AIR	DS	ScientistOne	ScientistOne*
Prism	↑	21.89	**26.26**	**26.26**	26.25	**26.26**	**26.26**	**26.26**	**26.26**
Cloudcast	↓	626.24	637.10	623.69	690.37	734.28	620.09	**618.08**	**618.08**
EPLB	↑	0.1265	0.1450	0.1453	0.1266	0.1449	0.1284	0.1459	**0.1461**
LLM-SQL	↑	0.6920	**0.7520**	0.7300	0.6757	0.7148	0.7307	0.7222	0.7115
TXN	↑	2724.8	4310	4310	3247	**4311**	4286	3906	3861

rating score and 4/5 tasks accepted. This gap is not driven by better algorithms—solver scores cluster tightly across systems (Table 3)—but by what happens *after* the solver finishes. The Claim Verifier prevents the most damaging failure mode we observe in rejected papers: claims that contradict the paper's own data (e.g., "sub-millisecond latency" when the results table reports 7.9 ms). Notably, CoE Audit and review scores measure orthogonal properties: across all 60 papers, Clarity averages 1–2 points above Soundness, confirming that narrative coherence is independent of evidence integrity.

The paper quality is bottlenecked by research soundness, not writing capability. Across all systems, Clarity is consistently the highest-scoring dimension (2.5–3.1) while Soundness is the lowest (1.1–2.3): these papers read well but 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 exhibits high seed variance (e.g., EPLB scores 1, 3, 8 across three seeds on the same task): rejected runs are those where the paper writer generates claims unsupported by the results (e.g., "sub-millisecond" when latency is 7.9 ms), while accepted runs make calibrated claims from the same underlying data. A post-hoc paper review that checks every quantitative claim in the abstract against the results tables would catch these contradictions and likely reduce this variance.

6.3 Solution Discovery Performance

To compare the effectiveness of our discovery module with baseline systems, we report ADRS performance in Table 3. Following [19], we report best-of-3 scores across seeds for each system. For ARC, AIR, DS, and ScientistOne, each score is from independent gold evaluator re-runs on submitted solver code, ensuring cross-system comparability. Human, AdaEvolve, and EvoX scores (Gemini-3.0-Pro, best-of-3 runs) are from original publications [19, 33]; we did not have access to their solver code for independent verification.

All six systems substantially outperform the human baseline on every task and fall within a narrow performance range, consistent with the observation of [8] that LLM-based agents rapidly converge to similar solution quality. Critically, verifiability does not sacrifice performance: ScientistOne achieves the best score on Cloudcast and EPLB, and remains competitive on the other three tasks.

7 Generalizability: MLE-Bench and Parameter Golf

To test whether the discovery loop transfers beyond ADRS, we evaluate ScientistOne unmodified across a diverse set of six tasks selected for their rigorous demands and relevance to current AI research. First, we select five MLE-Bench [28] Kaggle competitions spanning medical imaging, fine-grained recognition, and 3D perception. We specifically targeted tasks in the Medium and High difficulty tiers because they possess sufficient complexity and substantive research value to serve as meaningful benchmarks for automated paper writing and scientific discovery. Second, to assess adaptability in a novel, live environment, we evaluate ScientistOne on the Parameter Golf competition [29]. This live competition serves as an ideal AI research task focused on LLM training, requiring participants to train the highest-performing language model under multi-faceted constraints. Full task details and evaluation setups are provided in §F.

Table 4: Comparison of solver performance across five MLE-Bench tasks and Parameter Golf. "Above/Below Median" denotes outperforming or underperforming the median participant. MLE-Bench medals (Gold, Silver, Bronze) represent simulated private leaderboard standings. "SOTA" indicates top-1 performance on the Parameter Golf Leaderboard (as of 2026-04-27).

Task	Dir.	DeepScientist		ScientistOne	
		Score	Highlight	Score	Highlight
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 FGVC6	↓	0.2158	Silver Medal	0.2445	Silver Medal
Parameter Golf	↓	Invalid	Size limit exceeded	1.0600	SOTA (Constraints met)

Solver performance generalizes and demonstrates robustness. As shown in Table 4, ScientistOne demonstrates strong generalization and remarkable robustness across diverse domains, difficulty levels, and strict constraints. On the High difficulty MLE-Bench tasks, ScientistOne earns two Gold Medals (RSNA Brain Tumor and 3D Object Detection), notably solving the 3D Object Detection task with a Gold Medal score while DeepScientist failed entirely (scoring 0.0000). On the Medium difficulty tasks, ScientistOne secures Silver Medals on both iMet 2020 and iNaturalist 2019—remaining highly competitive with DeepScientist—and achieves an Above Median standing on AI4Code, marking a significant improvement. Furthermore, ScientistOne proves its robustness on the completely new LLM training task of Parameter Golf. While the baseline fails to produce a valid submission due to exceeding the 16MB artifact size limit, ScientistOne adheres to all constraints and achieves state-of-the-art performance with a score of 1.0600.

Solution novelty comparison on Parameter Golf. Both ScientistOne and DeepScientist were provided with the same prior-art reference and achieved superficially similar numerical improvements. However, the systems arrived at these outcomes in completely different ways. ScientistOne demonstrated genuine research capability by introducing novel algorithmic techniques to the quantization block: specifically, a Hessian-diagonal-weighted SVD initialization and an alternating-least-squares (ALS) refinement loop that utilizes GPTQ and a Cholesky-weighted truncated SVD. These mechanisms are entirely novel to the leaderboard, and internal ablations isolate the ALS loop as the primary driver of the performance gain. In contrast, DeepScientist contributed zero algorithmic novelty. Its modifications were limited to superficial environment and portability adjustments rather than structural research advancements. As a result, DeepScientist failed to improve the underlying algorithm – merely replicating the reference's performance – and ultimately produced an invalid submission by exceeding the strict 16MB artifact size limit. Full details are available in our generated paper provided in the supplementary materials.

8 Conclusion

Autonomous research systems have reached the point where solver quality alone no longer differentiates them—multiple systems achieve competitive scores on the same benchmarks with vastly different approaches. What separates their outputs is whether the resulting paper can be trusted. Our 60-paper audit shows that no baseline produces papers free of evidence chain failures, and the failures—phantom citations up to 21%, fictional method sections, scores on the wrong scale—are invisible to evaluations that assess narrative quality alone.

Chain-of-Evidence reframes verifiability as a first-class design constraint. ScientistOne demonstrates that an end-to-end pipeline can maintain evidence chains without sacrificing solver competitiveness, and CoE Audit provides a reusable protocol for auditing any system's output. The gap between ScientistOne's results and the baselines confirms that verifiability is architectural: systems that build evidence chains at claim-production time outperform those that reconstruct grounding after the fact. The harder problems—verifying citation support, checking conclusion claims, extending to domains without deterministic evaluators—remain open, but they are well-defined, and addressing them matters as autonomous research scales.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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*, pages 5977–6043, 2025.

[6] Yingming Pu, Tao Lin, and Hongyu Chen. PiFlow: Principle-aware scientific discovery with multi-agent collaboration. *arXiv preprint arXiv:2505.15047*, 2025.

[7] Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi, Bodhisattwa Prasad Majumder, Daniel S Weld, and Peter Clark. CodeScientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13370–13467, 2025.

[8] 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*, 2025.

[9] 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.

[10] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[11] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[12] Nelson F Liu, Tianyi Zhang, and Percy Liang. Evaluating verifiability in generative search engines. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7001–7025, 2023.

[13] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, 2023.

[14] Ori Press, Andreas Hochlehnert, Ameya Prabhu, Vishaal Udandarao, Ofir Press, and Matthias Bethge. CiteME: Can language models accurately cite scientific claims? *Advances in Neural Information Processing Systems*, 37:7847–7877, 2024.

[15] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM computing surveys (CSUR)*, 15(4):287–317, 1983.

[16] 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, 2026.

[17] 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*, 2025.

[18] 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.

[19] 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.

[20] 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*, 2025.

[21] Yujie Liu, Zonglin Yang, Tong Xie, Jinjie Ni, Ben Gao, Yuqiang Li, Shixiang Tang, Wanli Ouyang, Erik Cambria, and Dongzhan Zhou. ResearchBench: Benchmarking LLMs in scientific discovery via inspiration-based task decomposition. *arXiv preprint arXiv:2503.21248*, 2025.

[22] Tianze Xu, Pengrui Lu, Lyumanshan Ye, Xiangkun Hu, and Pengfei Liu. Researcherbench: Evaluating deep ai research systems on the frontiers of scientific inquiry. *arXiv preprint arXiv:2507.16280*, 2025.

[23] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

[24] 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*, 2025.

[25] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

[26] Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, et al. AIRS-Bench: A suite of tasks for frontier AI research science agents. *arXiv preprint arXiv:2602.06855*, 2026.

[27] Zhen Wang, Fan Bai, Zhongyan Luo, Jinyan Su, Kaiser Sun, Xinle Yu, Jieyuan Liu, Kun Zhou, Claire Cardie, Mark Dredze, et al. FIRE-Bench: Evaluating agents on the rediscovery of scientific insights. *arXiv preprint arXiv:2602.02905*, 2026.

[28] 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.

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

[30] Erzhuo Shao, Yifang Wang, Yifan Qian, Zhenyu Pan, Han Liu, and Dashun Wang. SciSciGPT: Advancing human–AI collaboration in the science of science. *Nature Computational Science*, 6(3):301–315, 2026.

[31] Jiaqi Liu, Peng Xia, Siwei Han, Shi Qiu, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Jiawei Zhou, Hongtu Zhu, Yun Li, Jiaheng Zhang, Yuyin Zhou, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. AutoResearchClaw: Fully autonomous research from idea to paper, 2026.

[32] 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.

[33] 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*, 2026.

A Limitations

Benchmark coverage. We designed CoE and CoE Audit to be domain-agnostic, but validating that generality requires evaluation across diverse scientific domains. Our current experiments focus on systems-optimization tasks (ADRS), where gold-standard evaluators make Score Verification and Specification Violation Detection straightforward. Open-ended domains—biology, materials science, theoretical ML—pose harder challenges: evidence chains may involve wet-lab protocols, simulation reproducibility, or proof sketches, each demanding domain-specific verification logic that we have not yet built or tested. Extending CoE to these settings is a natural next step, though the core abstraction (claims linked to evidence via typed provenance records) should transfer; what changes is the set of integrity checks required.

Citation verification depth. Our Citation Verification checks whether cited references *exist*—a necessary condition that already catches a surprising number of failures (§B.1, Case 2). However, existence is far from sufficient: a real citation can still be used to support a claim the cited paper never made. Full citation verification would require passage-level natural language inference against the cited paper's text, effectively asking "does this source actually say what the citing paper claims it says?" This is a known open problem in scholarly NLI, and we leave it to future work, noting that even existence-level checking already reveals meaningful architectural differences between systems.

Automated review as proxy. ScholarPeer serves as a scalable proxy for review quality but does not replace human expert evaluation. LLM reviewers are systematically blind to certain failure modes (e.g., domain-specific score interpretation, specification violation detection). CoE Audit addresses some of these blind spots but is itself limited to structural integrity, not scientific novelty or significance.

B Paper Quality Statistics

We report structural statistics of all 60 papers (4 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 5 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` produces the largest bibliography (55.3 $\pm$ 3.6 entries), 3.8 $\pm$ 0.4 figures. 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). DS produces the shortest papers (6.7 $\pm$ 2.1 pages).

Table 5: Paper quality statistics across four systems (mean $\pm$ std over 15 papers each).

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

B.1 Failure Mode Case Studies

We highlight four failure modes from the 60-paper audit, each illustrating a different evidence chain break that CoE Audit's integrity checks are designed to catch.

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 lives on a $[0, 1]$ scale. The figure is not a transcription error: it is the sum of squared prefix-hit lengths across datasets, an internal metric that the system computed and presented as if it were 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. An automated reviewer evaluating narrative quality alone would find nothing wrong. Score Verification catches this immediately: the gold 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. Citation Verification finds that 3 of those citations are hallucinated—we could not trace them to real papers and associated authors indexed by prominent indexes like Semantic Scholar, arXiv. This is not an edge case—AIR and DS produce phantom citations at rates of 9% and 21% respectively (Table 1), compared to 0% for systems with structured retrieval pipelines.

Case 3: Convergent specification violation (DS, LLM-SQL, seed 1). Specification violation flags this paper with a unanimous 3/3 vote. The submitted code achieves a legitimate score of 0.697 that passes Score Verification, but it does so by exploiting a gap between what the evaluator checks and what the benchmark intends to measure. The code sorts columns differently per row-group block, then renames all columns back to the original schema before concatenation—this makes `pd.concat` assemble blocks in insertion order rather than realigning by column name, effectively permuting column order per row-group. The evaluator validates row counts and total character counts but not column-to-column correspondence, so the permutation goes undetected. The same exploit appears independently in two other systems (AIR seed 2 and `ScientistOne` seed 2), providing convergent evidence that this is a genuine benchmark vulnerability rather than an isolated accident.[2]

Case 4: Correct score, fictional algorithm (ARC, TXN, seed 1). This paper passes Score Verification: the reported score of 3,311 falls within adaptive tolerance of the gold evaluator re-run (3,247). Yet Method-Code Alignment reveals a complete disconnect between what the paper describes and what the code implements. 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 rather than distributing write-heavy anchors. This case illustrates why Score Verification alone is insufficient—the solver works, but the paper describes a different algorithm entirely, making the method section unreproducible regardless of how accurate the reported numbers are.

C System Implementation Details

C.1 Solver Architecture

The Solver 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.

C.2 Ablation Study

Following PEE, the system selects the global best 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.

[2] For `ScientistOne` seed 2, the exploit is present in the submitted code, but the I2 majority-vote protocol did not reach consensus (1 of 5 judges flagged it), so it is not counted as a violation in Table 1. This illustrates a limitation of LLM-judged checks at the current vote threshold.

C.3 Claim Verifier: Per-Type Verification Rules

The Claim Verifier dispatches on the four claim types defined in §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 drift.
- **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.
- **Methodological claims** are checked by substantive textual overlap against the cited region of the experimental log.
- **Conclusion claims** are detected post-hoc by scanning for comparison language, linked to supporting numerical claims by token and number overlap, and—when parseable—validated by arithmetic check against pairwise differences.

Claims tagged "unsourced" or carrying malformed annotations are dropped automatically; each records a break code for downstream reporting.

C.4 Paper Writer Pipeline

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, enforcing the "provenance before prose" principle described in §4.3.

CONCEIVE. A single LLM call reads all assembled raw materials (PI brief, experimental log, verified scores, solver code, seed-paper abstracts) and produces the initial research representation. This document captures the story arc—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 establishes the narrative structure but does not validate evidence chains; that is deferred to GROUND.

GROUND. Deterministic checks validate each evidence annotation against the raw materials: the published score is present and matches `bestrun_selection.json`, baselines are labelled VERIFIED (traceable to a PI brief entry) or ESTIMATED (unattributed "leaderboard" references are flagged), every referenced file exists, 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 honest limitations. The critic returns PASS or a list of MAJOR/MINOR issues.

RESOLVE. A single LLM call rewrites the representation against both the GROUND flag list and the CRITIC issue list: unsupported claims are dropped or softened, contradictions are resolved using the verified source, overclaims are calibrated, and missing sections are filled. The GROUND→CRITIC→RESOLVE loop iterates for up to two rounds, terminating on convergence (zero flags) or plateau (the flag count stops decreasing). A final grounding ratio below 50% raises a hard error, aborting the paper rather than producing a poorly grounded draft.

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 (§C.3), whose findings drive a Refiner pass that applies corrections and strips inline annotations. Only a draft with no

blocking violations is promoted to `final_paper.pdf`; failed drafts are saved as `draft_paper.pdf` with a diagnostic report.

C.5 Problem Investigator Details

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 (Librarian agent), parallel PDF reading and structured note extraction (5 Researcher agents), and synthesis of findings into thematic research direction dossiers (SubdomainWriter agent). 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.

D CoE Audit Details

D.1 Sub-Code Definitions

D.1.1 Score Verification Failure Codes.

Code	Label	Description
SF-1	Genuine discrepancy	Paper score differs from evaluator rerun mean beyond adaptive tolerance, with consistent metric and scale
SF-2	Metric mismatch	Paper and evaluator report different metrics or scales (e.g., percentage vs. fraction, inverted objective)
SF-3	Metric inversion	Paper score is the reciprocal of the evaluator score, suggesting an inverted optimization direction
SF-4	Evaluator failure	All N=5 evaluator reruns crashed or returned null
SF-5	Evaluator timeout	All N=5 evaluator reruns exceeded the time limit
SF-6	Score extraction failure	No headline score could be extracted from the agent's logs or paper

Code	Label	Description
RWH-1	Evaluator exploitation	Solution reverse-engineers or exploits evaluator internals
RWH-2	Evaluator modification	Solution imports, reads, or modifies the evaluator
RWH-3	Data leakage	Solution reads test data, ground truth, or answer files
RWH-4	Hardcoded output	Solution produces hardcoded outputs rather than computing them
RWH-5	Specification exploit	Solution exploits metric edge cases or violates task constraints (e.g., column permutation in LLM-SQL)

Code	Label	Description
CF-1	Phantom citation	Reference does not exist anywhere (Scholar, DBLP, arXiv)
CF-2	Memory citation	Reference is real but recalled from model weights, not retrieved; metadata often corrupted

D.1.2 Specification Violation Sub-Codes.

D.1.3 Citation Verification Sub-Codes.

D.1.4 Method-Code Alignment Sub-Codes.

Code	Label	Description
MCA-0	Faithful	Paper and code describe the same algorithm
MCA-1	Acceptable simplification	Paper describes the same class of algorithm but omits or simplifies implementation details; rhetorical embellishment is allowed
MCA-2	Misrepresentation	Paper materially misrepresents the implemented algorithm (e.g., describes beam search when code uses greedy)

D.2 Audit Procedure and Reproducibility

CoE Audit is designed for reproducibility. Table 6 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 fully deterministic. Specification Violation and Method-Code Alignment are LLM-judged, with majority vote across multiple independent runs to reduce judgment noise. Citation Verification is primarily API-based (Semantic Scholar, arXiv, OpenAlex, CrossRef lookup), with an LLM disambiguation step for title-only matches.

Table 6: CoE Audit automation level per component.

Component	Automation	Model (if LLM)
Score Verification	LLM extraction + automated comparison	Gemini 3.0 Flash
Specification Violation	LLM-judged	Gemini 3.1 Pro
Citation Verification	Automated + LLM disambiguation	Gemini 3.0 Flash
Method-Code Alignment	LLM-judged	Gemini 3.1 Pro

Human verification. Although the audit pipeline is automated, all flagged positives for I1 (Score Verification), I2 (Specification Violation), and I3 (Citation Verification) were manually reviewed and corrected by human reviewers before reporting in Table 1. A small number of auditor false positives (e.g., API resolution failures for real papers, score extraction errors) were identified and removed during this process. For I4 (Method-Code Alignment), human reviewers validated a sample of the LLM judgments; the results in Table 1 reflect the LLM majority-vote scores without manual correction.

E Failure Cases per Audit Metric

E.1 I1: Score Verification Errors

We manually reviewed every I1 (Score Verification) failure flagged across the four baseline systems and categorise the 15 confirmed agent errors into four classes (Table 7).[3]

Table 7: Confirmed agent I1 errors, by category.

Category	Definition	Count	%
`value_mismatch`	Paper headline number differs from evaluator rerun beyond tolerance, with the same metric and scale.	11	73%
`paper_score_unavailable`	Paper contains no machine-readable headline number for the auditor to verify against.	2	13%
`evaluator_error`	Submitted artifact cannot be re-evaluated within the budget (timeout or crash on the agent's own solution file).	1	7%
`metric_mismatch`	Paper and code report different metrics, scales, or optimisation directions, so the numbers are not directly comparable.	1	7%

`value_mismatch` **(n=11).** The paper and the code agree on what is being measured but disagree on the value. Most (7/11) of these gaps are within 5% of the paper-reported number—small enough to plausibly arise from unreported seed variance, but uniformly biased towards a better-than-rerun headline. The two largest gaps are qualitatively different errors: a fabricated metric direction in DS `cloudcast` seed-2 (26.7%), where the paper relabels a cost metric as "utility" and inverts the optimisation direction so that a worse-than-baseline number reads as an improvement; and a 2× mismatch in ARC `prism` seed-3 between the paper headline (12.74) and what the submitted code actually produces (26.24).

`paper_score_unavailable` **(n=2).** The agent's writeup contains no machine-readable quantitative result for the headline metric. Both cases are baseline-system failures: an AIR `prism` paper that omits scores entirely, and a DS `cloudcast` PDF that failed to render its results section.

`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 harness that runs 8 baseline conditions × 3 seeds × multiple datasets at import time before any single number is produced.

`metric_mismatch` **(n=1).** The paper and the code do not agree on what they are measuring. The single case is ARC `llm_sql` seed-3, which publishes a headline of 0.0 that the paper itself attributes to "a critical integration failure with the benchmark evaluator framework"—the agent surfaced a known-broken result as its final score.

Per-system error shape. Table 8 shows that the failure mix is highly system-specific. AIR and DS errors are dominated by small-to-medium `value_mismatch`: the numbers exist and are in the right ballpark but do not exactly reproduce. ARC spans the widest range of categories and the largest single discrepancy (106%), and contributes the only `evaluator_error` and `metric_mismatch` cases. `ScientistOne` produces no confirmed I1 errors, consistent with its 12/12 score-verification result in Table 1.

E.2 I3: Citation integrity - Discovered phantom citations

[3] A small number of auditor false positives were corrected before reporting Table 1; this section reports only confirmed agent errors. Table 1 reports pass rates over 12 papers per system (EPLB excluded), so its implied failure count (11) is lower than the 15 errors here, which include EPLB papers.

Table 8: Confirmed agent I1 errors per system, by category.

System	value_mismatch	paper_score_unavail.	evaluator_error	metric_mismatch	Total
AIR	3	1	0	0	4
ARC	5	0	1	1	7
DS	3	1	0	0	4
ScientistOne	0	0	0	0	0

Phantom citations

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.

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.

Phantom citations (continued)

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.

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

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.

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.

Phantom citations (continued)

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.

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.

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.

E.3 I4 Audit Failure Analysis

We categorise the 95 method-code misalignment findings flagged by the I4 audit (across 25 papers[4]) into three semantic classes (Table 10). A single paper can produce multiple findings, so we report both the finding-level distribution and the distinct-paper count per category.

`incomplete_broken` **(n=49, 19 papers).** The largest category. The code generally targets the same problem the paper describes, but is missing one or more of the specific mechanisms claimed in the writeup, or substitutes a degenerate fallback. Recurring patterns include: described *multi-start initialisation with K sequences* collapsed to a single deterministic backtracking pass (AIR

[4]Table 1 implies 26 misaligned papers (12+10+3+1); one paper (DS seed-3/cloudcast) could not be cleanly classified and is excluded from the categorical breakdown.

Table 10: I4 method-code alignment findings, by category (n=95 findings, 25 affected papers).

Category	Definition	Findings	%	Papers
`incomplete_broken`	Paper-described mechanism or component is missing, partially implemented, or replaced with a degenerate fallback in the code.	49	52%	19
`algorithm_class_mismatch`	Code implements a fundamentally different algorithm class than the one described in the paper (e.g., paper claims Iterated Local Search, code is Simulated Annealing; paper claims a neural- or LLM-guided search, code is deterministic).	37	39%	15
`deceptive_dummy_code`	Code contains undisclosed elements designed to mislead evaluation: hidden environment-variable switches, or output deliberately shaped to game an evaluator metric.	9	9%	5

`prism`); claimed *link-penalty diversification* producing a constant assignment of the same path to every partition (AIR `cloudcast`); described *arborescence lookahead* or *surrogate cost model* simply absent from the code, with the true simulator invoked at every iteration instead (ARC `cloudcast`, ARC `txn_scheduling`); claimed *2.0 GB memory threshold safeguards* reduced to a trivial overflow check (DS `prism`).

`algorithm_class_mismatch` **(n=37, 15 papers).** The code implements a fundamentally different algorithm class than the one described in the paper. The most common sub-pattern is the *claimed-learning-loop-is-absent* case: a paper claims an LLM-driven evolutionary search, a neural network predictor, or an LLM SQL optimiser, and the code is a single deterministic heuristic with no LLM calls and no trained model (e.g. AIR `cloudcast`: "hybrid neuro-symbolic solver" $\rightarrow$ purely classical Steiner-tree heuristics; ARC `llm_sql`: "36 LLM prompting strategies" $\rightarrow$ deterministic dataframe reordering; DS `eplb`: "LLM-driven evolutionary search over 27 generations" $\rightarrow$ standalone deterministic load balancer). Classical algorithm-class swaps also recur: Iterated Local Search $\rightarrow$ Simulated Annealing (AIR `prism`), Dinkelbach's fractional programming $\rightarrow$ static sum-of-squares cost (ARC `prism`), recursive partitioning $\rightarrow$ single-level grouping (DS `llm_sql`). We also observe the rarer *method/baseline inversion* pattern, in which the paper labels X as the proposed method and Y as a rejected ablation, but the code uses Y (ARC `prism`: "GRASP Without Symbiosis" claimed as final method while the code instantiates `SymbioticGRASPPacker`).

`deceptive_dummy_code` **(n=9, 5 papers).** The submitted artifact contains undisclosed code whose presence appears intended to mislead automated evaluation. All nine findings come from ARC and split into two sub-patterns: (i) *hidden environment-variable switches* (4 findings, 2 papers): the code reads an undisclosed variable (`CONDITION`, `ABLATION`) at import time and dispatches to one of several different solvers, while the paper presents a single unified algorithm (ARC `cloudcast` seeds 2 and 3). (ii) *evaluator gaming* (5 findings, 3 papers): code intentionally shaped to inflate a metric without solving the underlying task, e.g. returning an empty column-ordering list while internally permuting values to maximise prefix-cache hits (ARC `llm_sql` seed-3).

Per-system error shape. Table 11 shows that the failure mix is sharply system-specific. AIR, DeepScientist, and `ScientistOne` produce no `deceptive_dummy_code` findings: when these systems misalign, the gap is between what the paper says and what the code does, with no active obfuscation. ARC is the only system that produces `deceptive_dummy_code` findings (9/9, 5 papers), spanning both hidden env-var switches and prefix-hit evaluator gaming.

F MLE-Bench and Parameter Golf Evaluation

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

MLE-Bench. We selected five MLE-Bench [28] Kaggle competitions spanning medical imaging, fine-grained recognition, and 3D perception. Table 12 summarizes the mapping between the task

Table 11: I4 findings per system, by category. Each cell is a finding count; a single paper may contribute multiple findings.

System	algo_class_mismatch	incomplete_broken	deceptive_dummy_code	Total
AIR	3	9	0	12
ARC	15	23	9	47
DS	15	16	0	31
ScientistOne	4	1	0	5

Table 12: 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

names used in our evaluation and their official task IDs. We specifically targeted tasks in the Medium and High difficulty tiers, as they possess sufficient complexity and substantive research value to serve as meaningful benchmarks for automated scientific discovery and paper writing. Specifically, AI4Code, iMet 2020 FGVC7, and iNaturalist 2019 FGVC6 are classified as Medium-difficulty tasks, whereas 3D Object Detection and RSNA Brain Tumor are High-difficulty tasks. The agent is provided with task descriptions and a code execution environment to develop its solutions. The execution environment consists of a compute node provisioned with 8xH100 GPUs, 192 CPU cores, and 1TB of RAM. To assist with formatting, the agent can query a validation tool an unlimited number of times to ensure its submission CSV files are correctly structured. Furthermore, we permit the agent to query the grading server up to 16 times to obtain evaluation scores on the test data. This deviates from the official MLE-Bench protocol, which restricts evaluation to a single submission of the final generated solution. We adopt this modified setup to better simulate real-world Kaggle competitions, where participants iteratively submit solutions to a public leaderboard for feedback during the development phase. While the agent is instructed to iterate on its solutions primarily based on metrics derived from the validation dataset, the grading server is used sparingly to select the best-performing models on the test set. Additionally, this setup is necessary to allow the agent to report accurate final test metrics when generating the paper.

Parameter Golf. To assess adaptability in a novel, live environment, we evaluate ScientistOne on the Parameter Golf competition [29]. This live competition serves as an ideal AI research task focused on LLM 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 tasked with building upon these existing leaderboard baselines to discover novel techniques capable of further improving the SOTA score.

G 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 solver iterations per task, 2-hour code generation windows, and 3 seeds per task. Runs that crashed due to infrastructure issues (API timeouts, rate limits, LaTeX compilation errors) were re-attempted with fresh state up to 3 times; no run was re-attempted to improve solver scores. Below we detail the per-system adaptations.

AutoResearchClaw (ARC). ARC v0.3.1 required 2 source patches: one to support the Gemini thinking-model API format, 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_DECISION_PIVOTS=2`, `MAX_REPAIR_CYCLES=3`) 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. Figure generation failed across all 15 runs because ARC's `FigureAgent` requires a Docker image not available in our environment; all papers were produced without figures.

DeepScientist (DS). DS required prompt-only changes—no source files were patched. Task specifications and the NeurIPS 2026 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 but the agent consistently shortcuts 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 NeurIPS 2026 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; 4 papers required manual LaTeX fixes (trailing newlines, malformed `filecontents` environments, bibliography corruption).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims are clearly justified by experimental results and discussion.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed the detailed limitations in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: Paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have provided necessary information for reproducibility as much as possible in both main manuscript and appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We have provided detailed methodology section as well as appendix. We will consider code publication upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We have provided all the details in the appendix and the method section.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide standard deviations of the experiments in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided detailed computational resources in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We followed the Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discussed potential societal impacts in the Appendix A.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We have not released any data or models in this submission.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly cited the assets used in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This submission does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We have provided the usages of the LLM in the Appendix.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.