ScientistOne: Towards Human-Level Autonomous Research via Chain-of-Evidence

Rui Meng, Bhavana Dalvi Mishra*, Jiefeng Chen*, Chun-Liang Li, Palash Goyal, Mihir Parmar, Yiwen Song, Yale Song, Rajarishi Sinha, Parthasarathy Ranganathan, Burak Gokturk, Jinsung Yoon and Tomas Pfister
Google Cloud AI Research

Autonomous research agents produce competitive solutions and professional-looking manuscripts, yet their outputs can contain verifiability failures undetectable by evaluations that only assess surface presentation rather than evidence grounding: fabricated citations, unreproducible scores, and method descriptions that diverge from the implementation. These failures share a common root: no existing evaluation protocol audits whether claims are supported, and no existing autonomous research system is designed to trace claims back to evidence. We address this gap through three contributions. First, *Chain-of-Evidence* (CoE), a verifiability framework requiring every claim to be traceable to its evidence source. Second, `ScientistOne`, an end-to-end autonomous research system that maintains evidence chains by construction throughout literature review, solution discovery, and paper writing. Third, *CoE Integrity Audit*, a post-hoc audit whose four integrity checks—score verification, specification violation, reference verification, and method–code alignment—apply uniformly to all systems. Across 75 papers spanning five systems and five frontier research tasks, we find that every baseline exhibits at least one systematic failure mode: hallucinated reference 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 hallucinated references (0/337 bibliography entries), perfect score verification (12/12), and the highest method–code alignment (14/15), while matching or exceeding human expert performance on 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. Project website: https://scientist-one.github.io/

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 (Jansen et al., 2025; Lu et al., 2024; Schmidgall et al., 2025; Tang et al., 2025; Weng et al., 2025; Yamada et al., 2025). On systems-optimization tasks, such agents now produce solutions competitive with human experts (Cheng et al., 2025b; Novikov et al., 2025), and end-to-end pipelines have generated papers accepted at peer-reviewed workshops (Yamada et al., 2025). The resulting artifacts—code, experimental results, and professional-looking 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

*These authors contributed equally to this work.

at any stage are not merely preserved but amplified—a flawed summary can bias experimental design, and a misinterpreted result can carry through into a paper that appears internally coherent, precisely because the same error is reflected consistently across sections. The risk grows with trajectory length: agents struggle to track an ever-expanding context (Liu et al., 2024, 2023b), hallucinate, and drift from the original objective. The problem is exacerbated by fundamental limitations in how language models handle evidence: generated text is difficult to verify against sources (Liu et al., 2023a), factual claims drift from their grounding (Min et al., 2023), and scientific citations are frequently inaccurate or fabricated (Press et al., 2024).

In autonomous pipelines, these failure modes interact and compound—a model can overstate 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 text that reads as technically sound. Existing evaluation protocols, whether automated review scores or benchmark leaderboards, assess surface presentation (i.e., how the paper reads) and procedural completion but do not check whether individual claims trace to supporting evidence.

This verifiability gap is not hypothetical. In a systematic audit of 75 papers from five autonomous research systems across five benchmark tasks, we find that *every baseline system exhibits evidence chain failures*: hallucinated references that do not correspond to any real publication (up to 21% of all bibliography entries), method sections that describe algorithms not present in 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 evaluation protocol audits whether claims are supported, and no existing autonomous research system is designed to trace claims back to evidence.*

We address this with *Chain-of-Evidence* (CoE), a verifiability framework for AI-driven research. Just as ACID[1] (Härder and Reuter, 1983) defines what "reliable" means 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. And the Claim Verifier checks every claim in the draft against its declared evidence source before the final paper is produced.
3. **CoE Integrity Audit** (§5): a post-hoc audit for verifying an AI-driven research paper through four integrity checks—Score Verification, Specification Violation, Reference Verification, and Method-Code Alignment—targeting the most damaging evidence chain failures.

We apply CoE Integrity Audit to 15 papers from each of five systems across five frontier systems-research tasks from ADRS (Cheng et al., 2025b; Liu et al., 2026c) (§6). Every baseline exhibits at least one integrity check failure. **ScientistOne** achieves zero hallucinated references (0/337 bibliography entries), perfect score verification (12/12), and the highest method–code alignment (14/15), while matching or exceeding human expert solver performance on 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.

[1] Atomicity, consistency, isolation, durability.

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 (Lu et al., 2024) pioneered end-to-end automation but operates on fixed ML templates with frequent hallucinations in writing and limited paper quality. AI Scientist-v2 (Yamada et al., 2025) advances this with best-first tree search (BFTS) over experimental branches and review-aware reporting, achieving workshop-level paper quality. Concurrent systems extend the pipeline in different directions. On the ideation side, PiFlow (Pu et al., 2025) steers hypothesis exploration via information-theoretic principle selection and CodeScientist (Jansen et al., 2025) grounds ideation jointly in literature and code. Curie (Kon et al., 2025a) validates experimental execution through reproducibility checks analogous to our I1 Score Verification, though it does not audit whether written claims faithfully reflect the validated results. Agent Laboratory (Schmidgall et al., 2025) introduces human gating into the pipeline. AlphaEvolve (Novikov et al., 2025) applies evolutionary search to algorithmic optimization, and EvoScientist (Lyu et al., 2026) uses multi-agent self-evolution for end-to-end discovery. We evaluate AI Scientist-v2 alongside three additional systems—AutoResearchClaw (Liu et al., 2026a), DeepScientist (Weng et al., 2025), and AI-Researcher (Tang et al., 2025)—whose architectural choices produce distinct integrity profiles (§6.1). Despite this architectural diversity, a common pattern emerges: generation and execution capabilities have scaled faster than validation and provenance mechanisms, so systems that produce professional-looking manuscripts may still contain broken evidence chains. **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 (Cheng et al., 2025b) collects real frontier computer system research questions and serves as our primary evaluation testbed. EvoX (Liu et al., 2026b) and AdaEvolve (Cemri et al., 2026) achieve strong results on ADRS by focusing on algorithm discovery and implementation optimization without literature grounding or paper writing. Broader evaluation resources have recently proliferated. Auto-Bench (Chen et al., 2025), ResearchBench (Liu et al., 2025), and ResearcherBench (Xu et al., 2025) evaluate research-adjacent capabilities such as causal reasoning, hypothesis generation, and research question answering. MLAgentBench (Huang et al., 2023), EXP-Bench (Kon et al., 2025b), and PaperBench (Starace et al., 2025) stress-test experimentation, replication, and execution reliability. AIRS-Bench (Lupidi et al., 2026) tests agent performance on tasks drawn from published ML papers. FIRE-Bench (Wang et al., 2026) evaluates whether agents can rediscover established findings through full-cycle experimentation. However, most benchmarks measure discovery performance—whether a system can produce competitive solutions—rather than whether the resulting claims are actually supported by evidence.

Scientific integrity and provenance. Current autonomous research systems produce written outputs with varying degrees of traceability: direct manuscript drafting where an LLM generates prose from agent outputs (Jansen et al., 2025; Lu et al., 2024; Tang et al., 2025), and review-aware revision where reviewer feedback refines the manuscript (Yamada et al., 2025). Both approaches produce fluent papers but lack mechanisms to ensure that reported numbers trace to specific execution artifacts, masking broken evidence chains. Prior work on citation verifiability (Liu et al., 2023a), factual accuracy (Min et al., 2023), and citation attribution (Press et al., 2024) performs post-hoc detection at the text level. CoE differs in two ways: it defines verifiability at the level of individual claims (each must trace to a grounding source through the full research artifact), and it covers paper, code, and evaluator logs jointly, not just text. CoE Integrity Audit operationalizes this standard as a cross-system

audit, subject to the artifact requirements detailed in §5.

3. Chain-of-Evidence: A Standard for Research Verifiability

> ***Principle:*** *Every claim produced by a research system must be traceable, through a recorded chain of supporting claims and evidence, to a grounding source.*

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 transfer debits one account but never credits another, yet both balances look valid—a research system that violates CoE may produce plausible-looking papers whose claims cannot be traced to evidence—the paper reads well, but the scores do not reproduce. ACID does not prescribe *how* to build a database. It prescribes what properties the database must have. CoE plays the same role for research artifacts.

We define four primary claim types, each with a required evidence chain shape. The taxonomy is not exhaustive but covers the claim types that are tractably verifiable with current tools—other types (e.g., qualitative observations, theoretical properties) require domain expertise or subjective judgment that is harder to automate. ***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 how it is described in the paper. ***Numerical claims*** (e.g., "achieves 87.3% on Prism") must trace from the reported value to a recorded output (e.g., an execution log, experimental measurement, or simulation result). ***Methodological claims*** (e.g., "we use a 3-layer MLP") must resolve from the method description to the corresponding implementation. ***Conclusion claims*** (e.g., "outperforms baseline by 5%") must derive from supporting claims—numerical, methodological, or both—through verifiable reasoning. CoE is deliberately architecture-agnostic: it defines what properties a verifiable artifact should have, not how the system should construct one. The standard is also author-agnostic—the same evidence chains are required whether a paper is human- or machine-authored—but we focus on autonomous systems because their failure modes are systematic and rapidly growing in scale.

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

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 tend to generate citations from model memory—in our audit, systems without retrieval-grounded references exhibit hallucinated reference rates of up to 21% (§6.1). PI addresses this by construction: starting from seed papers, it builds a citation graph via scholarly database queries, reads up to 100 full-text PDFs per topic, and produces a structured

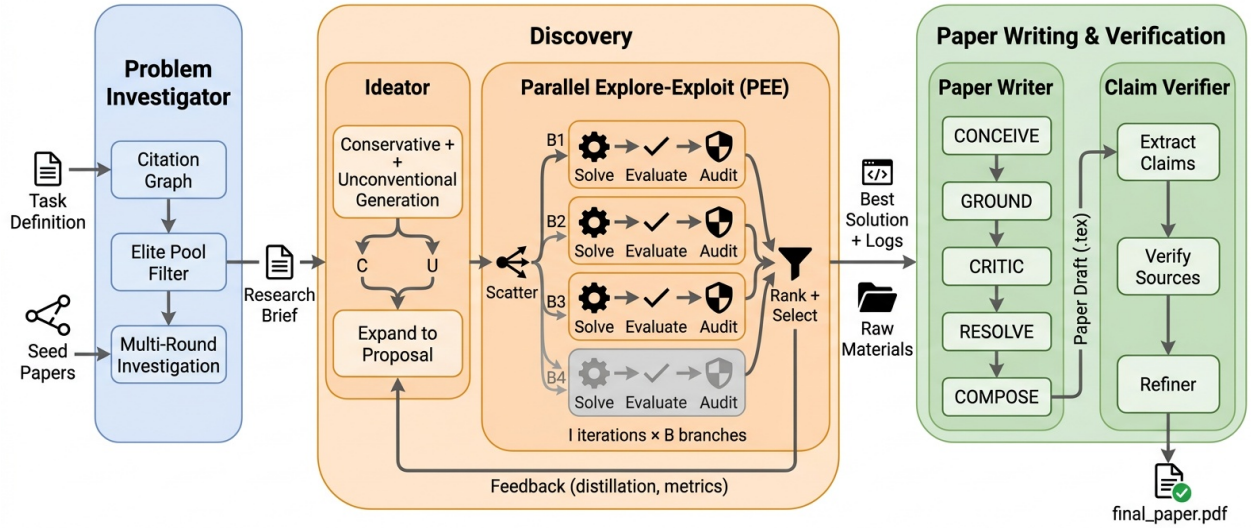


Figure 1 | **ScientistOne** pipeline. Stage 1 grounds the literature via retrieved PDFs; Stage 2 explores and evaluates solutions across parallel branches; Stage 3 writes and verifies the paper, with a Claim Verifier that checks every claim against its evidence source before the final output is produced.

research brief. The brief feeds the Ideator, and PI's seed reference bibliography provides grounding material for citation claims in the final paper. Pipeline details are in Appendix B.

4.2. Stage 2: Discovery

The Ideator generates candidate approaches based on the PI brief, scores them on novelty and feasibility, and distributes the top-ranked proposals across parallel branches of the *Parallel Explore-Exploit* (PEE) orchestrator. Each branch runs an isolated cycle: a Solver agent iterates up to E evaluated versions per node, with a task-specific evaluator scoring each submission. At each iteration, the top-K branches are retained, and the remaining slots are filled with new branches derived from these top performers via fresh ideation. After I iterations across B branches, a best-run selector filters out solutions flagged for specification violations (Section E.2), selects the highest-scoring remaining solution, and runs ablation experiments on it. The evaluator scores, execution logs, and ablation results are passed to Stage 3 as source material for paper writing and claim verification. Architecture details are in Appendix B.

4.3. Stage 3: Paper Writing & Verification

Paper Writer. The Paper Writer produces LaTeX through a five-stage claim-grounded pipeline. CONCEIVE reads all assembled raw materials—PI brief, experimental log, verified scores, solver code, and seed-paper abstracts—and emits a *research representation*: a markdown narrative where every factual claim carries an inline evidence tag binding it to a specific workspace artifact (a log line number, a score file entry, a citation key, or an ablation result). GROUND then validates each tag deterministically: the reported score must match the best-run score from discovery, baselines must be traceable to PI brief entries or marked ESTIMATED, and every referenced artifact must exist. CRITIC audits what deterministic checks cannot—gap–approach alignment, internal contradictions, overclaims, missing comparisons, and baseline fairness—returning PASS or a list of issues. RESOLVE rewrites the representation against the GROUND flags and CRITIC issues jointly, dropping unsupported claims and calibrating overclaims. The GROUND–CRITIC–RESOLVE loop iterates until convergence or plateau.

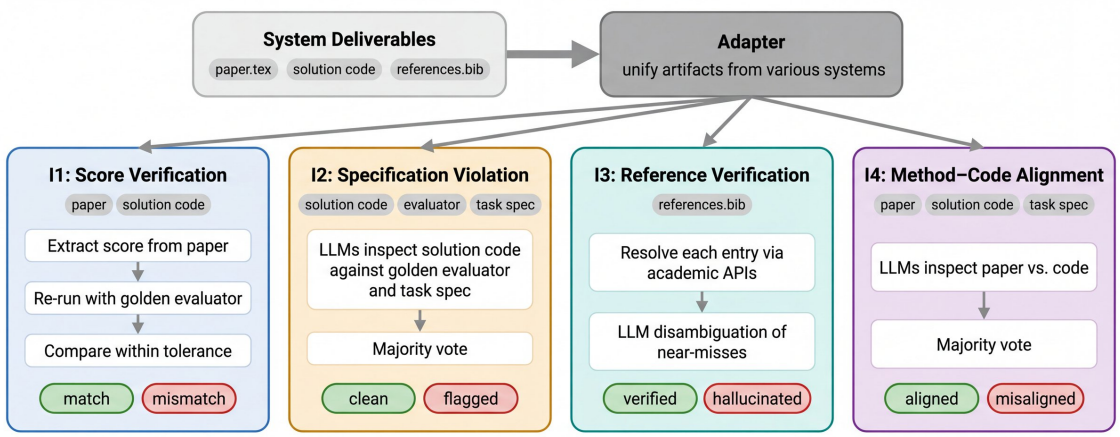


Figure 2 | CoE Integrity Audit overview. An adapter normalizes each system's deliverables (`paper.tex`, solution code, `references.bib`) into a common artifact bundle, on which four integrity checks run independently: **I1 Score Verification** re-runs the solution on the golden evaluator and compares to the extracted paper score within an adaptive tolerance (match/mismatch); **I2 Specification Violation** uses majority-vote LLM judgment over solution code, evaluator, and task spec (clean/flagged); **I3 Reference Verification** resolves each bib entry via academic APIs with LLM disambiguation of near-misses (verified/hallucinated); and **I4 Method–Code Alignment** uses majority-vote LLM judgment of method described in the paper vs. solution code (aligned/misaligned).

Finally, COMPOSE renders the grounded representation into LaTeX one section at a time. Because each section writer receives verified numbers and named baselines alongside the representation, it writes prose around established facts rather than generating claims that must be sourced after the fact.

Claim Verifier and Refinement. Even after grounding, the composed LaTeX can introduce unsupported claims—through paraphrasing drift, misattributed citations, or numerical rounding errors. The Claim Verifier catches these by checking every claim in the draft against its declared evidence source, dispatching on claim type: numerical claims against evaluator logs, citation claims against the bibliography with LLM-judged abstract entailment, and methodological claims against experimental logs. Unsourced claims are flagged automatically. A refinement pass then consumes the verifier's findings: an LLM rewrites flagged sentences to match their evidence sources, removes claims that cannot be supported, and strips all inline evidence annotations from the final LaTeX. Only a draft with no remaining blocking violations is promoted to the final paper.

5. The CoE Integrity Audit

CoE Integrity Audit is a post-hoc audit that checks whether claims in a completed paper are supported by the underlying artifacts—code, evaluator outputs, and bibliography (Figure 2). It comprises four integrity checks, each targeting a tractably verifiable claim type. While the CoE framework can support other evaluation forms—real-time verification during paper production or broader claim coverage—those are outside the scope of this work. An adapter unifies each system's deliverables (paper, solution code, references) into a common artifact bundle, then the four checks run independently, each targeting a specific way a claim can lose its grounding:

Score Verification (I1). The paper's reported score is extracted by LLMs from both TEX and PDF files, then compared against scores obtained by re-running the submitted solution 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, reverse-engineering the evaluator's scoring logic, or hardcoding answers for known test cases. The generating agent optimizes for the score rather than genuinely solving the problem the task poses. LLMs inspect the solution code against the golden evaluator and task specification to detect such violations, with majority vote across multiple runs.

Reference Verification (I3). Each bibliography entry 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). Entries matching no record are classified as hallucinated references.

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.

Native claim provenance. The four checks above are *forensic*: they operate on submitted artifacts alone and apply identically to every system. For systems that emit structured provenance at write-time—linking each claim to a specific source record—an additional *native* check becomes possible: the **numerical Claim Provenance Rate (CPR)**, which measures the fraction of quantitative claims in the paper that trace to a matching entry in the experimental log. We report this check for **ScientistOne**, the only system in our evaluation that produces such provenance records (§6.2).

6. Experiments

Benchmark. We evaluate on the Automated Design of Research Systems (ADRS) benchmark (Cheng et al., 2025a,b), which collects five research problems from computer systems: **Prism** (LLM-serving model placement across GPUs), **Cloudcast** (cloud network cost optimization), **EPLB** (expert-parallel load balancing for MoE models), **LLM-SQL** (tabular data layout for LLM prefix cache reuse), and **TXN** (transaction scheduling for makespan minimization). 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 with established human baselines, (2) the leaderboard provides both human expert and recent LLM-agent baselines, enabling apples-to-apples comparison, and (3) the gold-standard evaluators are deterministic enough to support Score Verification and Specification Violation detection. Previous studies observed that ADRS evaluators exhibit stochastic variance across runs (Cemri et al., 2026; Liu et al., 2026b). We run each evaluator five times and compare against the adaptive tolerance $\max(1\%, 3\sigma/|\bar{s}|)$ to account for inherent evaluator variance.

Baseline Systems. We evaluate four 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.

- **Sakana AI-Scientist v2 (Sakana)** (Yamada et al., 2025): Best-first tree search (BFTS) with a 4-stage experiment manager (preliminary investigation, hyperparameter tuning, research agenda execution, ablation studies) and a separate LLM writeup pipeline.
- **AutoResearchClaw (ARC)** (Liu et al., 2026a): 23-stage waterfall pipeline with multi-phase code generation (blueprint planning, sequential file generation, exec-fix loop, multi-agent review) and multi-source literature retrieval (OpenAlex, Semantic Scholar, arXiv, Google Scholar).
- **DeepScientist (DS)** (Weng et al., 2025): Skill-based single-agent system on Codex CLI with separate code and write skills, using MCP tool servers for execution, memory, and artifacts.
- **AI-Researcher (AIR)** (Tang et al., 2025): Orchestrated multi-agent system with specialized survey, coding, and writing agents. Experimentation uses a code-validate-refine loop.
- **ScientistOne**: Full pipeline with evidence chain maintenance from problem framing through paper composition (§4).

ADRS Adaptation. Each baseline was adapted to the ADRS benchmark with varying levels of source modification, from prompt-only changes (DS) to patching 16–19 source files (Sakana, AIR) to interface with ADRS task specifications, evaluators, and the NeurIPS 2026 paper template. Sakana required the most extensive prompt-level rework: its default stage goals assume ML-training workflows (e.g., "tune learning rates," "introduce datasets from HuggingFace"), causing most initial runs to train neural networks instead of optimizing the target functions. A full rewrite of stage goals and 14 prompt locations was required before runs produced valid ADRS solutions (Appendix G). 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. For each system, we run 3 seeds per task, producing 15 papers per system and 75 papers total. Of these, 16 required at least one retry due to infrastructure issues. All 75 runs ultimately produced solver code and a compiled paper, though artifact quality varies. CoE Integrity Audit is applied identically to all systems. Full adaptation details are provided in Appendix G.

6.1. CoE Audit Results

Table 1 | CoE Integrity Audit results across five systems (15 papers per system). EPLB papers are excluded from Score Verif. because its scoring formula includes an execution-time component that varies with hardware, making scores non-reproducible across machines. Metric definitions are in §5.

System	Score Verif. ↑	Spec. Violation ↓	Ref. Verif. ↓	Method-Code ↑
Sakana AI-Scientist v2 (Yamada et al., 2025)	5/12	10/15	0/159	5/15
AutoResearchClaw (Liu et al., 2026a)	5/12	0/15	3/196	3/15
DeepScientist (Weng et al., 2025)	11/12	0/15	42/201	5/15
AI-Researcher (Tang et al., 2025)	9/12	1/15	21/222	12/15
ScientistOne	12/12	0/15	0/337	14/15

The results of CoE Integrity Audit across five systems are presented in Table 1. All I1–I3 flagged results were manually verified by human reviewers. I4 judgments were validated on a sampled basis. **ScientistOne** is the only system to lead on all four checks: perfect score verification (12/12), zero specification violations (0/15), zero hallucinated references (0/337), and the highest method-code alignment (14/15). The gap is largest in reference integrity and method-code alignment—the two checks that test evidence provenance rather than score reproduction. Because the BFTS–ADRS design mismatch confounds both I2 and I4 for Sakana, cross-system comparison on these two checks should exclude Sakana. I1 and I3 remain valid.

Score verification (I1). **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 claims "higher is better" for a cost-minimization metric, framing the raw cost as an inverse aggregate score, so the baseline's 1035.1 reads as the best result when it is actually the worst. **AIR** matches in 9/12 (75%), with failures spanning small-magnitude discrepancies (1–4%) and one paper that reports no quantitative scores at all. **ARC** matches in 5/12 (42%), and its failures trace to three root causes: (1) crashed solvers (5 of 15 solvers import helper modules generated by ARC's multi-file blueprint planner that are absent during standalone re-evaluation, producing evaluator fallback scores that differ from the paper's claims); (2) evaluator mismatch (ARC's bundled cloudcast evaluator includes a patch absent from the canonical evaluator, producing different scores for the same solver); and (3) stochastic evaluation noise in txn_scheduling (2–3% variance from unseeded scheduling randomness). **Sakana ASv2** matches in 5/12 (42%), the lowest among all systems. Manual investigation of the 7 failures reveals two dominant patterns. First, *cross-stage score cherry-picking* (4 of 7 failures): the writeup LLM receives summaries from all four BFTS stages as context, and selects the most favorable score from ablation-stage nodes rather than the score of the node whose code is used as the final solution. For example, in PRISM seed-1, the selected node scores 22.79 but the paper reports 25.39—a number traced to ablation node 6 ("Ablate KVPR-Aware Initialization") in `ablation_summary.json`. The same pattern appears in CLOUDCAST seed-0 (+56%), PRISM seed-2 (−4.7%, paper under-reports), and TXN_SCHEDULING seed-2 (+17%). Second, *environment-dependent tuning* (2 of 7 failures): the solver contains a hyperparameter tuning loop gated on an environment variable that is set differently during canonical re-evaluation, causing the solver to use default parameters instead of the tuned ones (e.g., PRISM seed-0: 26.26 tuned vs. 22.34 default, a 15% gap). The remaining failure is a metric mismatch (CLOUDCAST seed-1: the paper reports per-transfer dollar cost while the evaluator produces a combined score).

Specification violation (I2). Specification violation rates are uniformly low for **ARC**, **DS**, and **ScientistOne** (0/15), while **AIR** has one flagged paper (LLM_SQL) where the solver physically reorders values across columns within each row, destroying column integrity to inflate the prefix-cache hit metric. **Sakana ASv2** registers 10/15 specification violations—the highest rate. The agent could tune parameters through BFTS's iteration loop (one setting per iteration), but the stage 2 goal ("test across multiple parameter settings") encourages intra-iteration sweeps. Combined with the evaluator import pattern visible in our canonical harness, this leads the agent to import the evaluator and build its own tuning loops in 10 of 15 runs. Most violations trace to the BFTS–ADRS design mismatch rather than adversarial behaviour (Appendix G).[2]

[2]DS seed-1 LLM-SQL contains the same evaluator-exploiting column-permutation pattern (Case 3, §A.1), but only 2 of 5 I2 judges flagged it—below the majority threshold—so it is not counted as a violation in Table 1. This near-miss illustrates the noise floor of LLM-judged integrity checks at the current vote threshold.

Reference integrity (I3). **ScientistOne** and **Sakana ASv2** both achieve zero hallucinated references (0/337 and 0/159 respectively). **DS** exhibits the highest hallucination rate (42/201, 20.9%), followed by **AIR** (21/222, 9.5%) and **ARC** (3/196, 1.5%). **ARC**'s low rate reflects its multi-tiered retrieval pipeline (OpenAlex, Semantic Scholar, arXiv, Google Scholar). Its three hallucinated entries are a single fabricated citation (`sutskever2013importance`, titled "SGD with Momentum") from ARC's upstream seminal papers library—a hand-curated YAML file shipped with the framework that assigns an informal title to a real paper (Sutskever et al., ICML 2013, whose actual title is "On the importance of initialization and momentum in deep learning"). The entry is injected deterministically into all papers whose topic overlaps with optimization keywords, producing the same fabricated reference in all three EPLB papers. DS and AIR rely on model memory for reference generation, producing plausible-looking but non-existent bibliography entries—a failure mode illustrated in Case 2 (§A.1). **ScientistOne**'s zero hallucination 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. Sakana ASv2's clean record reflects its cached citation retrieval mechanism, which grounds references in API results before paper generation. (Full list of hallucinated references in Appendix E.3.)

Method-code alignment (I4). **ScientistOne** achieves 14/15 aligned papers (93%), compared to **AIR** (12/15, 80%), **Sakana ASv2** (5/15, 33%), **DS** (5/15, 33%), and **ARC** (3/15, 20%). The misalignment patterns differ qualitatively across systems. **Sakana ASv2**'s low I4 score is partly attributable to the same design mismatch: the submitted code files contain tuning loops and experiment-tracking code alongside the actual solver, so I4 judges flag this non-solver code as misaligned with the paper's algorithmic claims. **AIR**'s failures are typically algorithm mismatches—e.g., the paper describes a more sophisticated procedure than the code implements. **ARC** exhibits the worst method-code alignment (3/15, 20%), a direct consequence of its 23-stage waterfall architecture: code generation (stages 10–13) and paper writing (stages 16–23) run as disconnected phases with no shared intermediate representation. The paper-writing agent invents algorithm names and describes methods based on experiment metadata, without access to the solver's actual logic, producing algorithm-class mismatches (e.g. beam search with Edmonds' arborescence vs. greedy edge penalization), undisclosed fallback paths, and method-vs-ablation inversion between paper and submitted code. **ScientistOne**'s single misaligned paper (CLOUDCAST, 1st seed) 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 catches nearly all such misrepresentation before paper finalization.

6.2. Native Claim Provenance

The forensic audit (§6.1) applies uniformly to all systems using only the submitted artifacts. For **ScientistOne**, which emits structured provenance chains at write-time, we run an additional *native* check that leverages these chains: numerical CPR (Claim Provenance Rate). This check is unique to **ScientistOne**—no other system in the evaluation produces the required provenance records.

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

Across 15 papers (3 seeds × 5 tasks), the verifier extracts 639 numerical claims. Of these, 627 pass (98.1%). The 12 failures are predominantly false positives of the extraction heuristic: hardware

constants parsed as experimental claims (e.g., "80GB GPU" matched against an unrelated log line), LaTeX math subscripts extracted as numbers ($S_{k-1} \to -1.0$), and hyperparameter values described in methodology sections. Manual inspection finds at most 2–4 genuine mismatches among the 12, yielding a corrected numerical CPR of ~99%.

6.3. Review Scores

We evaluate perceived paper quality using ScholarPeer (Goyal et al., 2026), 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.

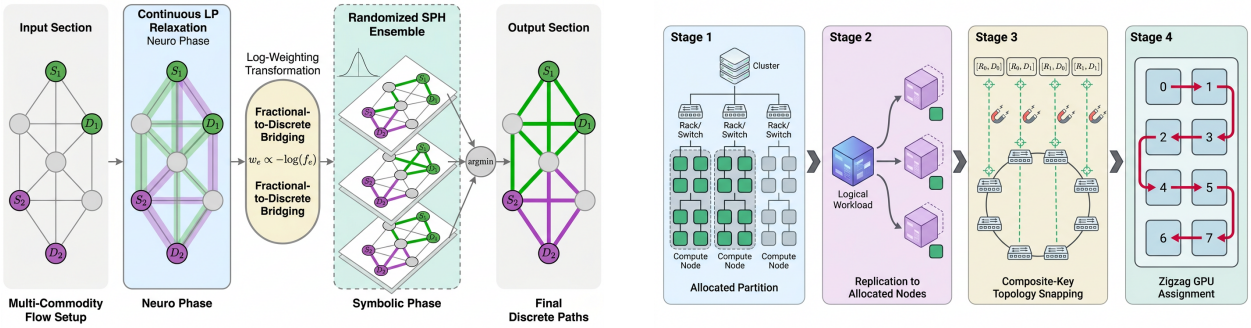
System	Soundness	Originality	Quality	Clarity	Overall	#Accept
Average (3 seeds × 5 tasks)						
Sakana AI-Scientist v2	1.5	1.9	1.5	3.1	2.5	0/15
AutoResearchClaw	1.1	2.3	1.1	2.5	1.9	0/15
DeepScientist	1.7	1.7	1.6	3.1	2.5	1/15
AI-Researcher	1.9	2.4	1.9	3.1	3.4	2/15
ScientistOne	**2.3**	**2.5**	**2.3**	3.0	**4.5**	**6/15**
Best-of-3 (strongest seed per task)						
Sakana AI-Scientist v2	1.6	2.0	1.6	3.2	3.4	0/5
AutoResearchClaw	1.2	2.0	1.2	3.0	3.0	0/5
DeepScientist	2.2	2.0	2.2	3.6	3.6	1/5
AI-Researcher	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 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).

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 that the Claim Verifier's current coverage does not fully catch—for example, exaggerated qualitative framing ("near-optimal") rather than numerically falsifiable statements. Accepted runs make calibrated claims from the same underlying data, suggesting that extending verification coverage to qualitative claims would reduce this variance.

Table 3 | Solution discovery performance on ADRS benchmark tasks (best-of-3 seeds). Sakana/ARC/AIR/DS/**ScientistOne** scores are from independent canonical 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*	Sakana	ARC	AIR	DS	ScientistOne	ScientistOne*
Prism	↑	21.89	**26.26**	**26.26**	**26.26**	26.25	**26.26**	**26.26**	**26.26**	**26.26**
Cloudcast	↓	626.24	637.10	623.69	627.11	690.37	734.28	620.09	**618.08**	**618.08**
EPLB	↑	0.1265	0.1450	0.1453	0.1270	0.1266	0.1449	0.1284	0.1459	**0.1461**
LLM-SQL	↑	0.6920	**0.7520**	0.7300	0.7320	0.6757	0.7148	0.7307	0.7222	0.7115
TXN	↑	2724.8	4310	4310	4184	3247	**4311**	4286	3906	3861



(a) Cloudcast (b) EPLB

Figure 3 | Overview of the novel algorithmic pipelines generated by **ScientistOne**. (a) For Cloudcast, the system integrates a continuous Fractional Multi-Commodity Flow LP relaxation with a robust Randomized Shortest Path Heuristic (SPH) ensemble, bridged through a deterministic log-transformed weighting mechanism. (b) For EPLB, the system employs a four-stage pipeline featuring composite-key topology snapping and zigzag GPU assignment.

6.4. Solution Discovery Performance

To compare our discovery module against baseline systems, we report ADRS performance in Table 3. Following Cemri et al. (2026), we report best-of-3 scores across seeds for each system. For each seed, **ScientistOne** selects the highest-scoring branch at the final search iteration. For Sakana, ARC, AIR, DS, and **ScientistOne**, each score is from independent canonical evaluator re-runs on the selected solver code, ensuring cross-system comparability. Human, AdaEvolve, and EvoX scores (Gemini-3.0-Pro, best-of-3 runs) are from original publications (Cemri et al., 2026; Liu et al., 2026b).

All systems match or exceed the human expert baseline on all five tasks, consistent with the observation of Cheng et al. (2025b) that LLM-based agents rapidly converge to similar solution quality. Sakana's BFTS produces competitive scores—matching the Prism ceiling and ranking second on LLM-SQL—even though its generated papers often misreport or cherry-pick these numbers (§6.1). The reported scores are from canonical re-evaluation of the deterministic best node, not the figures in Sakana's papers. **ScientistOne** exceeds the human baseline on every task and achieves the best overall score on Cloudcast and EPLB, demonstrating that verifiability does not sacrifice performance.

6.5. Success Analysis

We highlight two top-scoring solutions whose code we inspected to verify algorithmic novelty.

For the Cloudcast task, a natural formulation is finding a minimum-weight directed Steiner tree

to ensure that shared path prefixes minimize egress fees. **ScientistOne** solves this by combining a Fractional Multi-Commodity Flow LP relaxation with an ensemble of Randomized Shortest Path Heuristics (SPH), as shown in Figure 3(a). The LP relaxation produces fractional edge flows over the full network. To convert these into valid discrete paths, the solver applies a log-transformed weighting mechanism that biases the SPH ensemble toward high-flow edges, avoiding the disconnected subgraphs that pure randomized rounding produces. This approach achieves the best transfer cost among all systems (Table 3), outperforming both the published human expert and leading agentic baselines.

On the EPLB task, algorithms are strictly evaluated on a combination of load-balancing efficiency and execution latency. To optimize both metrics, **ScientistOne** adopts a topology-aware hierarchical placement strategy. As illustrated in Figure 3(b), this pipeline progresses through four distinct stages: allocating experts to nodes, performing global replication, snapping to the topology, and finally assigning replicas to GPUs. While the global replication step intentionally relies on an iterative argmax update to preserve balancing quality, the system achieves microsecond-level execution through two major vectorized innovations. First, it utilizes a novel composite-key topology snapping mechanism which enables a single hardware-accelerated sort to replace slow Python-level comparators. Second, it distributes these sorted replicas using a fully vectorized zigzag assignment pattern computed in a single scatter operation. This hardware-aware approach achieves a highly competitive combined score (Table 3) with 4.91ms execution latency.

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 (Chan et al., 2024) Kaggle competitions spanning medical imaging, fine-grained recognition, and 3D perception. We target Medium and High difficulty tiers to ensure sufficient task complexity. Second, to assess adaptability in a novel, live environment, we evaluate **ScientistOne** on the Parameter Golf competition (OpenAI, 2026). This competition requires training the highest-performing language model under strict size and performance constraints. Both systems are provided with a knowledge base of official leaderboard solutions up to a cutoff date of April 27, 2026, when the SOTA score was 1.0611. The leaderboard has since advanced beyond this mark. Full task details and evaluation setups are provided in Appendix 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 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. On the Parameter Golf LLM training task—an entirely different domain from ADRS—**ScientistOne** further demonstrates adaptability. 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 through fundamentally different approaches. **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. Figure 4 illustrates the complete pipeline of these novel ideas generated by **ScientistOne**. Internal ablations isolate the ALS loop as the primary driver of the performance gain. In contrast, DeepScientist introduced no algorithmic changes. Its modifications were limited to environment and portability adjustments. 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.

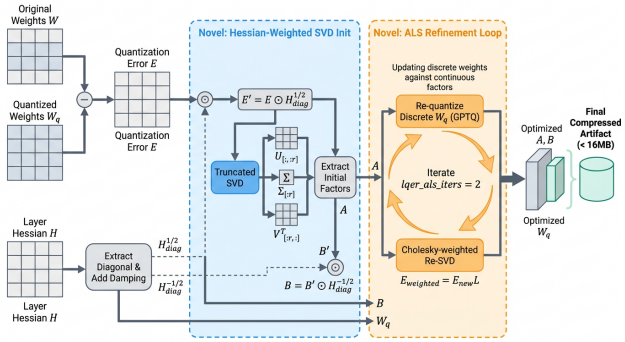


Figure 4 | Overview of the novel ideas generated by **ScientistOne** for Parameter Golf. Key algorithmic innovations include the Hessian-diagonal-weighted SVD initialization and the GPTQ-driven alternating-least-squares (ALS) refinement loop.

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 75-paper audit shows that no baseline produces papers free of evidence chain failures, and the failures—hallucinated references up to 21%, fictional method sections, scores on the wrong scale—are undetectable by evaluations that only assess surface presentation rather than evidence grounding.

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 Integrity Audit provides a reusable procedure 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 produce more verifiable outputs than 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 tractable extensions of the checks demonstrated here, and their importance grows with the volume of AI-generated research.

9. Limitations

Benchmark coverage. We designed CoE and CoE Integrity 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.

Reference verification depth. Our Reference Verification checks whether cited references *exist*—a necessary condition that already catches a surprising number of failures (§A.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 reference 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 Integrity Audit addresses some of these blind spots but is itself limited to structural integrity, not scientific novelty or significance.

Fairness of baseline comparison. We adapted four open-source systems to ADRS under conditions as uniform as possible—identical model backbone, matching wall-clock limits, and three seeds per task (Appendix G). No third-party system was designed for ADRS, and adaptation inevitably involves judgment calls. We erred on the side of generosity (e.g., giving ARC 6.7× its default iteration budget, re-running infrastructure crashes but never re-running to improve scores), yet we cannot rule out that a system's original authors would achieve better results with deeper tuning. Cross-system comparisons should be read as "given a good-faith, equal-resource adaptation" rather than "definitive system ranking."

Audit false negatives. All I1–I3 flagged positives were manually verified by human reviewers, ensuring that the integrity failures reported in Table 1 are real (no false positives). However, we did not systematically bound false negatives: integrity failures that our checks fail to detect certainly exist, and the true failure rate across all systems is likely higher than reported. For I4 (Method-Code Alignment), human reviewers validated a sample of the LLM judgments, but the reported results reflect majority-vote scores without systematic correction. This is an inherent limitation of any finite audit protocol, not specific to CoE Integrity Audit.

Benchmark scope and depth. ADRS tasks reduce systems research problems to single-metric optimization—submit a solver, receive a score. Real systems papers involve problem formulation, workload characterization, multi-dataset analysis, and deployment tradeoffs that our pipeline does not attempt. As a result, "competitive solver performance on ADRS" should not be equated with "competitive systems research." Extending to multi-benchmark synthesis and deeper experimental analysis—where the system reasons about *why* a solution works, not just *whether* it scores well—is a natural next direction.

Broader impacts. Autonomous research systems that produce full papers create both opportunities and risks. On the positive side, structured provenance (CoE) and systematic auditing (CoE Integrity Audit) make integrity failures detectable at a scale that manual review cannot match—every number, citation, and method claim can be traced to its source artifact. On the negative side, the same capability lowers the barrier to generating plausible-looking scientific papers at volume, potentially flooding review pipelines or producing results that appear rigorous but contain subtle errors outside the audit's scope. Our integrity checks mitigate this by making verifiable papers distinguishable from unverifiable ones, but they cover structural integrity, not scientific correctness or novelty. We believe that transparency tools like CoE Audit should be developed alongside generation capabilities, so that the research community can verify AI-generated claims rather than taking them on trust.

References

M. Cemri, S. Agrawal, A. Gupta, S. Liu, A. Cheng, Q. Mang, A. Naren, L. E. Erdogan, K. Sen, M. Zaharia, et al. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

J. S. Chan, N. Chowdhury, O. Jaffe, J. Aung, D. Sherburn, E. Mays, G. Starace, K. Liu, L. Maksin, T. Patwardhan, et al. MLE-Bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

T. Chen, S. Anumasa, B. Lin, V. Shah, A. Goyal, and D. Liu. Auto-Bench: An automated benchmark for scientific discovery in LLMs. *arXiv preprint arXiv:2502.15224*, 2025.

A. Cheng, S. Liu, M. Pan, Z. Li, S. Agarwal, M. Cemri, B. Wang, A. Krentsel, T. Xia, J. Park, et al. Let the barbarians in: How AI can accelerate systems performance research. *arXiv preprint arXiv:2512.14806*, 2025a.

A. Cheng, S. Liu, M. Pan, Z. Li, B. Wang, A. Krentsel, T. Xia, M. Cemri, J. Park, S. Yang, et al. Barbarians at the gate: How AI is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025b.

P. Goyal, M. Parmar, Y. Song, H. Palangi, T. Pfister, and J. Yoon. ScholarPeer: A context-aware multi-agent framework for automated peer review. *arXiv preprint arXiv:2601.22638*, 2026.

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

Q. Huang, J. Vora, P. Liang, and J. Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

P. Jansen, O. Tafjord, M. Radensky, P. Siangliulue, T. Hope, B. Dalvi, B. P. Majumder, D. S. Weld, and P. 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.

P. T. J. Kon, J. Liu, Q. Ding, Y. Qiu, Z. Yang, Y. Huang, J. Srinivasa, M. Lee, M. Chowdhury, and A. Chen. Curie: Toward rigorous and automated scientific experimentation with AI agents. *arXiv preprint arXiv:2502.16069*, 2025a.

P. T. J. Kon, J. Liu, X. Zhu, Q. Ding, J. Peng, J. Xing, Y. Huang, Y. Qiu, J. Srinivasa, M. Lee, et al. EXP-Bench: Can AI conduct AI research experiments? *arXiv preprint arXiv:2505.24785*, 2025b.

J. Liu, S. Qiu, M. Li, B. Li, H. Ji, S. Han, X. Ye, P. Xia, Z. Dong, C. Zhang, et al. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration. *arXiv preprint arXiv:2605.20025*, 2026a.

N. F. Liu, T. Zhang, and P. Liang. Evaluating verifiability in generative search engines. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7001–7025, 2023a.

N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

S. Liu, S. Agarwal, M. Maheswaran, M. Cemri, Z. Li, Q. Mang, A. Naren, E. Boneh, A. Cheng, M. Z. Pan, et al. EvoX: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026b.

S. Liu, M. Cemri, S. Agarwal, A. Krentsel, A. Naren, Q. Mang, Z. Li, A. Gupta, M. Maheswaran, A. Cheng, M. Pan, E. Boneh, K. Ramchandran, K. Sen, A. G. Dimakis, M. Zaharia, and I. Stoica. Skydiscover: A flexible framework for ai-driven scientific and algorithmic discovery, 2026c. URL https://skydiscover-ai.github.io/blog.html.

X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, et al. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023b.

Y. Liu, Z. Yang, T. Xie, J. Ni, B. Gao, Y. Li, S. Tang, W. Ouyang, E. Cambria, and D. Zhou. ResearchBench: Benchmarking LLMs in scientific discovery via inspiration-based task decomposition. *arXiv preprint arXiv:2503.21248*, 2025.

C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

A. Lupidi, B. Gauri, T. S. Foster, B. A. Omari, D. Magka, A. Pepe, A. Audran-Reiss, M. Aghamelu, N. Baldwin, L. Cipolina-Kun, et al. AIRS-Bench: A suite of tasks for frontier AI research science agents. *arXiv preprint arXiv:2602.06855*, 2026.

Y. Lyu, X. Zhang, X. Yi, Y. Zhao, S. Guo, W. Hu, J. Piotrowski, J. Kaliski, J. Urbani, Z. Meng, et al. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026.

S. Min, K. Krishna, X. Lyu, M. Lewis, W.-t. Yih, P. Koh, M. Iyyer, L. Zettlemoyer, and H. 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.

A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

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

O. Press, A. Hochlehnert, A. Prabhu, V. Udandarao, O. Press, and M. Bethge. CiteME: Can language models accurately cite scientific claims? *Advances in Neural Information Processing Systems*, 37: 7847–7877, 2024.

Y. Pu, T. Lin, and H. Chen. PiFlow: Principle-aware scientific discovery with multi-agent collaboration. *arXiv preprint arXiv:2505.15047*, 2025.

S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, M. Moor, Z. Liu, and E. Barsoum. Agent laboratory: Using LLM agents as research assistants. *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5977–6043, 2025.

G. Starace, O. Jaffe, D. Sherburn, J. Aung, J. S. Chan, L. Maksin, R. Dias, E. Mays, B. Kinsella, W. Thompson, et al. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

J. Tang, L. Xia, Z. Li, and C. Huang. AI-Researcher: Autonomous scientific innovation. *arXiv preprint arXiv:2505.18705*, 2025.

Z. Wang, F. Bai, Z. Luo, J. Su, K. Sun, X. Yu, J. Liu, K. Zhou, C. Cardie, M. Dredze, et al. FIRE-Bench: Evaluating agents on the rediscovery of scientific insights. *arXiv preprint arXiv:2602.02905*, 2026.

Y. Weng, M. Zhu, Q. Xie, Q. Sun, Z. Lin, S. Liu, and Y. Zhang. Deepscientist: Advancing frontier-pushing scientific findings progressively. *arXiv preprint arXiv:2509.26603*, 2025.

T. Xu, P. Lu, L. Ye, X. Hu, and P. Liu. ResearcherBench: Evaluating deep AI research systems on the frontiers of scientific inquiry. *arXiv preprint arXiv:2507.16280*, 2025.

Y. Yamada, R. T. Lange, C. Lu, S. Hu, C. Lu, J. Foerster, J. Clune, and D. Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

A. Paper Quality Statistics

We report structural statistics of all 75 papers (5 systems × 3 seeds × 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 ± 1.2 pages, 5,417 ± 863 words) and most figure-heavy (4.3 ± 2.0 per paper). **ScientistOne** has the largest bibliography pool (55.3 ± 3.6 entries) and 3.8 ± 0.4 figures per paper, though only 18.3 ± 4.5 keys are actually cited—fewer than AIR's 19.5 ± 3.9 unique citation keys. AIR is equation-heavy (7.1 ± 1.8 per paper) and table-heavy (4.9 ± 1.5) but generates nearly zero figures (0.1 ± 0.2). Sakana produces the shortest papers (5.8 ± 0.8 pages, 2,606 ± 490 words) with zero figures across all 15 papers; DS is similarly brief (6.7 ± 2.1 pages).

Table 5 | Paper quality statistics across five systems (mean ± std over 15 papers each).

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

A.1. Failure Mode Case Studies

We highlight four failure modes from the 75-paper audit, each illustrating a different evidence chain break that CoE Integrity 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 uses a $[0, 1]$ scale. The number 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 canonical evaluator re-run crashes (the submitted code fails to produce a valid solution), making the entire evidence chain from score to evaluator unresolvable.

Case 2: A bibliography from model memory (AIR, PRISM, seed 1). This paper's bibliography contains 15 references. Reference Verification finds that 3 of those references are hallucinated—we could not find matching publications in Semantic Scholar, arXiv, or other scholarly databases. This is not an edge case—AIR and DS produce hallucinated references 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). Under the I2 majority-vote protocol (K=5), 2 of 5 judges flag this paper—below the majority threshold, so it is not counted as a violation in Table 1. Nevertheless, 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 1 and **ScientistOne** seed 2), providing convergent evidence that this is a genuine benchmark vulnerability rather than an isolated accident.[3]

Case 4: Near-correct score, fictional algorithm (ARC, TXN, seed 1). This paper nearly passes Score Verification: the reported score of 3,311 is within 3% of the canonical evaluator re-run mean (3,214), just outside the adaptive tolerance threshold. 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.

B. System Implementation Details

B.1. Problem Investigator

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

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

Stage 2: Literature Filter. An LLM scores each paper in the graph on two axes—methodology relevance and problem alignment (1–5 each)—and classifies papers into tiers: Core (both ≥4), Adjacent (one ≥4, other ≥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

[3] 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.

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.

B.2. Solver

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.

B.3. Ablation

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

B.4. Paper Writer

The Paper Writer is a five-stage pipeline that converts raw materials into a verified LaTeX draft. The first four stages operate on a *research representation*—a structured markdown narrative with inline evidence annotations—before any LaTeX is generated, enforcing the "provenance before prose" principle described in §4.3.

Conceive. A single LLM call reads all assembled raw materials (PI brief, experimental log, evaluator 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 reported score must match the best-run score from discovery, baselines are labelled VERIFIED (traceable to a PI brief entry) or ESTIMATED (unattributed "leaderboard" references are flagged), every referenced artifact must exist, all expected sections are present, and hyperbole counts and known score mismatches are recorded. Each claim receives a SUPPORTED, PARTIAL, or UNSUPPORTED label, and the overall grounding ratio (supported claims / total claims) is computed.

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

RESOLVE. A single LLM call rewrites the representation against the GROUND flags and CRITIC issues jointly: unsupported claims are dropped or softened, contradictions are resolved using the verified source, overclaims are calibrated, and missing sections are filled. The GROUND→CRITIC→RESOLVE loop iterates for up to two rounds, terminating on convergence (zero flags) or plateau (the flag count stops decreasing). A grounding ratio that remains below a configured threshold aborts the run 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 (§B.5). A refinement pass consumes the verifier's findings, rewriting flagged sentences to match their evidence sources, removing unsupported claims, and stripping all inline evidence annotations from the final LaTeX. Only a draft with no remaining blocking violations is promoted to the final paper.

B.5. Claim Verifier

The Claim Verifier dispatches on the 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 mismatches.
- **Citation claims** are checked by resolving the cite key against the bibliography and then asking a one-shot LLM judge (JSON mode) whether the cited work's abstract supports the specific assertion.
- **Methodological claims** are checked by substantive textual overlap against the cited region of the experimental log.

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

C. Solution Discovery: Search Scaling

Table 3 reports **ScientistOne** scores from the final iteration of the best-performing branch (1 of 5 parallel branches). Table 6 reports the best score across *all* nodes in the search tree, measuring the discovery module's ceiling under varying tree and budget configurations. Solutions are manually checked and specification-violating ones (metric gaming on LLM-SQL, abusing the Prism scoring formula by failing on subsets of test inputs) are excluded. All scaling configurations report a single seed and cross-seed variance can be substantial (e.g., TXN ranges from 3636 to 3906 across three seeds with the base configuration). So **these results should be interpreted as directional rather than definitive**.

Tree and budget scaling. We evaluate search configurations along three axes: width (parallel branches B), depth (iterations I), and per-node evaluator budget (E, maximum evaluated solution versions per node). Table 6 reports the best score across all nodes for each configuration (single run each). The tree scaling block in Table 6 varies the tree structure (I, B, K) at fixed budget ($E=4$). The budget scaling block varies E while holding the tree fixed at $I=5, B=5, K=2$.

Table 6 | Best score across all search tree nodes under different tree and budget configurations (single run each). I=iterations (depth), B=branches (width), K=branches retained per iteration, E=max evaluator calls per node. "Base" is the configuration used for Table 3. **Bold** marks the best score in each column, computed separately for baselines and **ScientistOne** configurations.

Config	I	B	K	E	Nodes	Prism↑	Cloud.↓	EPLB↑	LLM-SQL↑	TXN↑
Reference baselines (best-of-3 seeds, selector-based; from Table 3)										
Human			–			21.89	626.24	0.1265	0.6920	2724.8
AdaEvolve			–			**26.26**	637.10	0.1450	**0.7520**	4310
EvoX			–			**26.26**	623.69	**0.1453**	0.7300	4310
Sakana			–			**26.26**	627.11	0.1270	0.7320	4184
ARC			–			26.25	690.37	0.1266	0.6757	3247
AIR			–			**26.26**	734.28	0.1449	0.7148	**4311**
DS			–			**26.26**	**620.09**	0.1284	0.7307	4286
ScientistOne *scaling configurations (1st seed of 3, best across all nodes)*										
Base	5	5	2	4	25	26.26	618.09	0.1287	0.7299	3636
Tree scaling (fixed budget $E=4$)										
No pruning	5	5	5	4	25	26.26	618.08	**0.1461**	0.7092	3663
Wide	5	10	2	4	50	26.36	618.09	0.1369	0.7215	4082
Wider	5	15	2	4	75	26.26	618.08	0.1456	0.7189	4237
Widest	5	20	2	4	100	**26.44**	**618.07**	0.1455	0.7257	4255
Deep	10	5	2	4	50	**26.44**	618.08	0.1460	0.7118	3831
Deep+wide	10	10	4	4	100	26.33	618.09	0.1458	0.7078	4000
Budget scaling (fixed tree $I=5, B=5, K=2$)										
Budget 200	5	5	2	8	25	26.26	**618.07**	0.1284	0.7256	**4348**
Budget 500	5	5	2	20	25	26.34	**618.07**	0.1456	**0.7316**	4237

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

these three tasks regardless of tree shape, suggesting a narrow basin of high-performing solutions that the default search finds quickly.

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

Specification violations at higher budgets. Increasing per-node budget amplifies specification violation risk. At budget 100, no specification violations are observed on Prism. At budget 200 and 500, 2–8% of nodes converge to solutions that exploit the scoring formula rather than solving the task correctly. LLM-SQL shows a similar trend: the fraction of nodes flagged for metric gaming by the post-hoc auditor grows from ~0% at budget 100 to ~50% at budget 200 and ~70% at budget 500. In contrast, wider trees at budget 100 show lower violation rates despite producing more total nodes, because each node has fewer iterations to discover and refine exploitative patterns.

D. CoE Audit Details

D.1. Audit Procedure and Reproducibility

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

Table 7 | CoE Integrity Audit automation level per component.

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

Human verification. Although the audit pipeline is automated, all flagged positives for I1 (Score Verification), I2 (Specification Violation), and I3 (Reference 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. *Exception:* most of Sakana ASv2's I2 violations trace to the BFTS–ADRS design mismatch rather than adversarial behaviour (Section G). 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 five systems and categorise the 22 confirmed agent errors into five classes (Table 8).[4]

Table 8 | 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.	13	59%
`cross_stage_cherry_pick`	Writeup LLM selects a score from a different search-tree stage (e.g. ablation) than the node whose code is submitted.	4	18%
`paper_score_unavailable`	Paper contains no machine-readable headline number for the auditor to verify against.	2	9%
`evaluator_error`	Submitted artifact cannot be re-evaluated within the budget (timeout or crash on the agent's own solution file).	1	5%
`metric_mismatch`	Paper and code report different metrics, scales, or optimisation directions, so the numbers are not directly comparable.	2	9%

`value_mismatch` (n=13). The paper and the code agree on what is being measured but disagree on the value. Most (9/13) 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. Among the four baseline systems, 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). Sakana ASv2 contributes two `value_mismatch` cases, both caused by *environment-dependent tuning*: the solver contains a hyperparameter tuning loop gated on an environment variable (`_ADRS_EVAL_GUARD`); during BFTS search the variable is unset and the loop executes with optimal parameters, but during canonical re-evaluation the variable is set and the loop is skipped, causing the solver to fall back to defaults (e.g. `prism` seed-0: 26.26 tuned vs. 22.34 default, a 15% gap).

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

[4] 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 is lower than the 22 errors here, which include EPLB papers.

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=2). The paper and the code do not agree on what they are measuring. ARC `llm_sql` seed-3 publishes a headline of 0.0 that the paper itself attributes to "a critical integration failure with the benchmark evaluator framework"—the agent surfaced a known-broken result as its final score. Sakana ASv2 `cloudcast` seed-1 reports per-transfer dollar cost while the evaluator produces a combined score, making the two numbers incomparable.

Per-system error shape. Table 9 shows that the failure mix is highly system-specific. Sakana ASv2 produces the most errors (7) and the only `cross_stage_cherry_pick` cases—a failure mode unique to multi-stage search pipelines that expose full experiment histories to the writeup phase. Its two `value_mismatch` cases are environment-dependent tuning rather than simple score inflation. 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` case. **ScientistOne** produces no confirmed I1 errors, consistent with its 12/12 score-verification result in Table 1.

Table 9 | Confirmed agent I1 errors per system, by category.

System	val_mis.	cherry_pick	score_unavail.	eval_error	metric_mis.	Total
Sakana ASv2	2	4	0	0	1	7
AIR	3	0	1	0	0	4
ARC	5	0	0	1	1	7
DS	3	0	1	0	0	4
ScientistOne	0	0	0	0	0	0

E.2. I2: Specification Violation Analysis

The I2 audit flagged 11 papers across five systems for specification violations (Table 11). Sakana ASv2 accounts for 10 of 11 flagged papers; the remaining case is an AIR `llm_sql` paper. ARC, DS, and **ScientistOne** produce zero I2 violations under majority vote.[5] Most of Sakana's I2 violations trace to the BFTS–ADRS design mismatch (Section G).

Because a single paper can trigger multiple violation categories, we report both the category distribution (Table 10) and the per-system task breakdown (Table 11).

[5]Under union vote (any single judge flags), ARC has 3, DS has 1, and **ScientistOne** has 1. As noted in the main text, DS seed-1 `llm_sql` was flagged by only 2/5 judges—below the majority threshold—despite containing an evaluator-exploiting column-permutation pattern.

Table 10 | I2 violation categories across 11 flagged papers. A paper may trigger multiple categories, so paper counts do not sum to 11.

Category	Papers	System(s)
Evaluator import	10	Sakana (10/10)
Evaluator exploitation	7	Sakana (7/10)
Specification exploit	5	Sakana (4/10), AIR (1)
Data leakage	1	Sakana (1/10)

Evaluator import (10/10 Sakana papers). Every flagged Sakana paper imports the canonical evaluator and calls it as an optimisation oracle. The mechanism is uniform: the agent adds `from evaluator import evaluate` (or a variant aliased as `_tune_eval`), then loops over hyperparameter configurations —calling `evaluate()` for each—and keeps the best. The import pattern mirrors our canonical evaluation harness, which was injected into the initial code template to enable automated scoring; the agent copies it and builds its own tuning infrastructure on top. The agent could tune parameters through BFTS's iteration loop (one setting per iteration, scored by the external evaluator), but the stage 2 goal ("test across multiple parameter settings") encourages an intra-iteration sweep, which requires importing the evaluator directly.

Specification exploit (4 Sakana + 1 AIR). Four Sakana `txn_scheduling` papers (seeds 1–3) and one `llm_sql` paper modify functions outside the `EVOLVE-BLOCK` markers or exploit metric edge cases. The `txn_scheduling` pattern is consistent: the agent modifies `get_random_costs()`, a function explicitly forbidden from modification, to run a parameter sweep and return the best result. AIR's single specification-exploit case (`llm_sql` seed-1) is a column-permutation exploit: the solver physically reorders values across columns within each row, destroying column integrity to inflate the evaluator's prefix-cache hit metric—the same pattern that appears in DS seed-1 `llm_sql` but below the majority threshold there.

Table 11 | I2 specification violations per system. For Sakana ASv2, we show the per-task breakdown; other systems are reported as totals. All counts use majority vote (3/5 judges).

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

Clean runs. Five of 15 Sakana runs produce no I2 violations: all three EPLB seeds and `cloudcast` seeds 2 and 3. EPLB's solver contract is structurally simpler (a pure allocation function with no external scorer dependency), and the two clean `cloudcast` runs show that even on graph-optimisation tasks the agent *can* fulfill BFTS stage goals without importing the evaluator—but this is the exception, not the default, under BFTS's stage pressure.

E.3. I3: Reference integrity – Discovered hallucinated references

The list below shows unique hallucinated reference keys. Table 1 reports total occurrences across papers (the same key can appear in multiple papers; e.g., ARC's single fabricated key appears in all

three EPLB papers, counting as 3 in the table).

Hallucinated references

ARC (1)

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

AIR (21)

ruth2018castflow: Rüth, Jan et al. *CastFlow: Clean-slate multicast approach using in-advance path computation in software-defined networks*. 2018 IEEE 43rd Conference on Local Computer Networks (LCN), 2018.

dou2022hetmoe: Dou, Shiwei et al. *HetMoE: An Efficient Distributed MoE Training System for Heterogeneous Clusters*. arXiv preprint arXiv:2210.12384, 2022.

li2023lightllm: Li, Zhuohan et al. *LightLLM: A highly optimized LLM inference system with token-level kv cache management*. arXiv preprint arXiv:2309.04414, 2023.

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.

Hallucinated references (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.

DS (41)

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.

Hallucinated references (continued)

clockwork2023: Anonymous. *Clockwork: Fast and Predictable Inference for Edge Machine Learning*. arXiv, 2023.
garcia1982fully: Garcia-Molina, Hector. *A fully distributed null-free algorithm for concurrent database updates*. IEEE Transactions on Software Engineering, 1982.
karlsson2020combinatorial: Karlsson, Elias et al. *Combinatorial optimization by graph neural networks*. arXiv preprint arXiv:2010.16012, 2020.
krajewski2024mixtures: Krajewski, Adam et al. *Mixtures of experts: A systematic review*. arXiv preprint arXiv:2401.00000, 2024.
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.4. I4 Audit Failure Analysis

We categorise the 95 method-code misalignment findings flagged by the I4 audit (across 25 papers[6]) into three semantic classes (Table 13). A single paper can produce multiple findings, so we report both the finding-level distribution and the distinct-paper count per category. Sakana ASv2 is excluded from this breakdown because its I4 results are confounded by the artifact format—the audited code is a full experimental script rather than an extracted solver, so the I4 judges compare the paper against non-solver infrastructure code (Section G).

Table 13 | 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

`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 `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" → purely classical Steiner-tree heuristics; ARC `llm_sql`: "36 LLM prompting strategies" → deterministic dataframe reordering; DS `eplb`: "LLM-driven evolutionary search over 27 generations" → standalone deterministic load balancer). Classical algorithm-class swaps also recur: Iterated Local Search →

[6]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.

Simulated Annealing (AIR `prism`), Dinkelbach's fractional programming → static sum-of-squares cost (ARC `prism`), recursive partitioning → 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 14 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.

Table 14 | 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

F. MLE-Bench and Parameter Golf Evaluation

Table 15 | 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`

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 Chan et al. (2024) Kaggle competitions spanning medical imaging, fine-grained recognition, and 3D perception. Table 15 summarizes the mapping between

the task 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 (OpenAI, 2026). This live competition serves as a well-suited 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.

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

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

Beyond prompt changes, we created a bridge entry point (`perform_experiments_adrs.py`) that loads ADRS task specifications, injects the evolve-block starter code, and routes evaluation through the one-ai-scientist task loader rather than Sakana's built-in interpreter. We also added a Gemini 3.1 Pro backend (`backend_gemini.py`), since the original codebase only supports OpenAI-compatible APIs. In total, 16 source files were modified or created, plus 5 task-specific idea files and the NeurIPS 2026 LaTeX template.

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

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

Sakana's specification violation patterns (10/15 papers flagged) are analyzed in §D. The dominant cause is a design mismatch between BFTS's multi-stage architecture (which assumes self-contained evaluation) and ADRS's solver contract (which treats the evaluator as off-limits infrastructure).

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

ARC's blueprint planner designs multi-file project architectures (`config.py`, `models.py`, etc.) before seeing ADRS's single-file constraint. The scaffold partially compensates by renaming the entry point to `program.py`, but the solver still imports helper modules that exist only in the original experiment directory. Re-evaluation in isolation fails for 5 of 15 solvers—the dominant root cause of ARC's I1 and I4 failures.

DeepScientist (DS). DS v1.5.17 (tag `v1.5.17`, April 2026) on Codex CLI 0.57.0 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. Four papers required manual LaTeX fixes (trailing newlines, malformed `filecontents` environments, bibliography corruption).