

SCI-AGENT-EXAM: Benchmarking Agents on Scientific Research Tasks

Tianyun Zhong^{1,2,*}, Wei Wang³, Jialin Li³, Jianhong Tu⁴, Liyao Li^{5,*}, Chuqiao Kuang⁴,
Tianshi Zheng^{6,*}, Songchi Zhou^{7,*}, Weiqi Zhai³, Qile Jin³, Jiajia Li³, Jiasheng Zheng^{1,2},
Boxi Cao², Yaojie Lu², Hongyu Lin², Shiwei Ye¹, Bing Zhao^{3,†}, Hu Wei³, Lin Qu³,
Xianpei Han^{2,†}

¹University of Chinese Academy of Sciences

²Institute of Software, Chinese Academy of Sciences

³AI Data, Alibaba Group ⁴Qwen Team, Alibaba Group

⁵Zhejiang University ⁶The Hong Kong University of Science and Technology

⁷Tsinghua University

*Work done during an internship at Alibaba, †Corresponding authors

zhongtianyun24@mails.ucas.ac.cn, xiongdiao@alibaba-inc.com, xianpei@iscas.ac.cn

Abstract

Large language model (LLM) agents are increasingly capable of supporting scientific workflows, but their capabilities across diverse research activities remain difficult to assess systematically. Existing benchmarks often focus on selected scientific capabilities or task types, limiting a comprehensive characterization of scientific agents' strengths and weaknesses in realistic research workflows. We introduce SCI-AGENT-EXAM, a workflow-oriented benchmark comprising 120 expert-curated tasks across 11 scientific fields and 7 research task types defined by the SCIENTIFIC WORKFLOW TASK TAXONOMY (SWTT). Each task requires both executable artifacts (e.g., code and scripts) and textual artifacts (e.g., reports and explanations). To evaluate the heterogeneous task outputs, we develop a hybrid evaluation protocol combining deterministic programmatic verification for executable artifacts with a calibrated, expert-validated LLM-based judge for textual artifacts. Beyond outcome assessment, we analyze progress and solution stability along execution trajectories to characterize how agents reach their final results. Experiments with 10 frontier LLM agents reveal substantial performance gaps: GPT-6 Astra achieves the highest overall score of 80.9, yet only 26.0% of its runs exceed the 0.95 success threshold. Further analyses uncover distinct capability profiles across scientific task types and substantial variation in execution efficiency and stability. Our code and data are available at <https://github.com/SKYLENAGE-AI/Sci-Agent-Exam>.

1 Introduction

Large language models (LLMs) have evolved from systems that primarily generate and transform text into agents capable of multi-step reasoning, tool use, code execution, and interaction with external envi-

ronments [7, 30, 42]. These capabilities enable LLM agents to move beyond answering individual scientific questions and participate in broader scientific workflows. Each scientific workflow task specifies a scientific objective, required deliverables, and evaluation criteria, enabling assessment under controlled

execution conditions.

Evaluating scientific agents thus requires examining what they produce, whether they can execute and explain their analyses, and how reliably they reach the results. This raises a central question:

What can scientific agents deliver across diverse research activities, and how efficient and stable is their execution and presentation?

Existing benchmarks have made substantial progress in evaluating scientific agents across diverse settings, including scientific question answering [36], scientific discovery [24] and analysis [7], research coding [15, 31], paper replication [32], project-level research [45], research in a single field [19] and executable scientific workflows [5, 35].

However, as summarized in Figure 1 and detailed in Table 1, two limitations remain in existing benchmarks. First, their explicit coverage of scientific activities remains uneven, with some activities only implicitly involved, partially covered, or not covered. This limits a systematic characterization of agents’ strengths and weaknesses across research activities. Second, their coverage of task deliverables remains incomplete. Scientific work not only requires agents to reach answers through executable results but also requires them to explain their analyses in writing. Yet the benchmarks compared in Table 1 do not consistently require and jointly evaluate all three forms of deliverables (answers, executable artifacts, and textual artifacts) for every task. Beyond these two limitations, final-answer scores alone provide limited insight into execution efficiency and solution stability: they do not reveal how many steps agents take to reach a result or how solution quality changes during execution. A workflow-oriented evaluation should therefore combine systematic coverage of scientific activities with comprehensive assessment of task deliverables and analysis of execution dynamics.

To address these evaluation needs, we introduce SCI-AGENT-EXAM, a workflow-oriented benchmark for evaluating LLM agents in scientific research activities. First, SCI-AGENT-EXAM proposes the SCIENTIFIC WORKFLOW TASK TAXONOMY (SWTT), which organizes scientific work into seven task types: Literature Synthesis, Hypothesis Validation, Tool Orchestration, Code Generation, Error Diagnosis, End-to-End Analysis, and Result Interpretation. The benchmark comprises 120 tasks spanning 11 scientific domains, with each task assigned to one task type and constructed and verified by researchers with relevant domain expertise. This taxonomy-guided

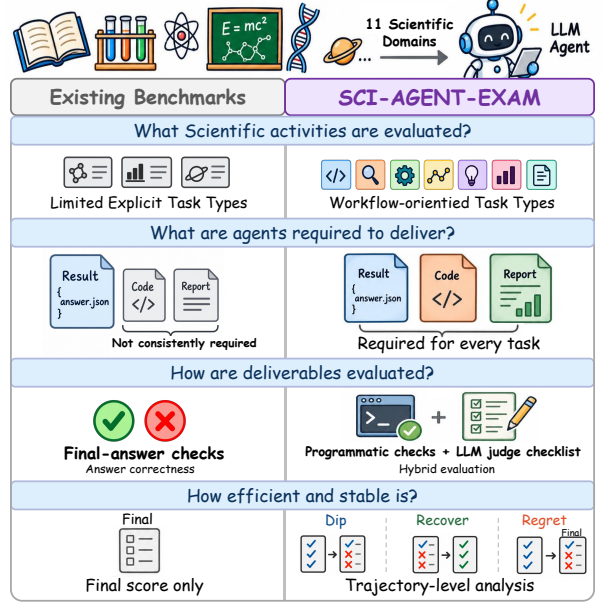


Figure 1 SCI-AGENT-EXAM versus existing benchmarks: workflow oriented task types, full deliverables (answer, code, report) per task, hybrid evaluation, with analysis for efficiency and stability .

design enables systematic comparison of agents’ capabilities across different scientific activities. Second, every task requires three forms of deliverables: a final structured answer, executable artifacts (e.g., code and scripts), and textual artifacts (e.g., reports and explanations). We assess these outputs using a hybrid protocol that combines deterministic programmatic verification with rubric-based LLM judgments. We additionally analyze intermediate workspace snapshots to characterize agents’ execution efficiency and solution stability, including patterns of regression and recovery.

We evaluate 10 frontier LLM agents on SCI-AGENT-EXAM and observe substantial differences in both overall performance and task-specific capability profiles. GPT-6 Astra achieves the highest overall score of 80.9, outperforming the second-ranked Claude Fable 5 by 13.6 points. Nevertheless, only 26.0% of its runs achieve a score of at least 0.95, indicating substantial room for improvement even for the strongest agent. Performance also varies considerably across scientific task types, revealing distinct strengths and weaknesses. Trajectory analysis further shows that stronger agents generally reach high-quality solutions earlier and undergo fewer regressions during execution.

Our main contributions are summarized as follows:

Benchmark	Task Setting	Task-Type Coverage							Required Deliverables			Scale	
		1	2	3	4	5	6	7	Answer	Exec.	Report	# Task	# Domain
SciBench	Scientific QA	○	○	○	○	○	○	●	●	○	○	869	3
DiscoveryBench	Scientific discovery	○	●	●	●	○	○	●	●	○	○	264	6
ScienceAgentBench	Scientific analysis	○	○	●	●	●	●	●	●	●	○	102	4
CORE-Bench	Computational reproduction	○	○	●	●	●	●	●	●	○	○	270	3
PaperBench	Paper replication	●	○	●	●	●	●	●	○	●	○	20	1
ResearchCodeBench	Research coding	○	○	●	●	○	●	○	○	●	○	212	1
SWE-bench Science	Scientific issue resolution	○	○	●	●	○	●	○	○	●	○	119	20
ASI-Bench	Project-level research	●	●	●	●	●	●	●	●	●	●	60	11
Terminal-Bench-Science	Scientific Tasks	○	●	●	●	○	○	●	●	●	●	70	5
SCI-AGENT-EXAM	Workflow-oriented tasks	●	●	●	●	●	●	●	●	●	●	120	11

Table 1 Comparison with representative scientific-agent benchmarks. **Task-Type Coverage** maps seven task types: ① Literature Synthesis, ② Hypothesis Validation, ③ Tool Orchestration, ④ Code Generation, ⑤ End-to-End Analysis, ⑥ Error Diagnosis, and ⑦ Result Interpretation. For task-type coverage, ● = explicitly targeted, ○ = involved but not distinct, and ○ = not covered. For **Required Deliverables**, ● = required in every task, ○ = required in some tasks, and ○ = not required. *Answer*: short answer/result; *Exec.*: code/artifact; *Report*: written report. Domain counts follow each benchmark’s categorization.

- We introduce SCI-AGENT-EXAM, a workflow-oriented benchmark comprising 120 tasks across 11 scientific fields and 7 research task types defined by SWTT, with each task constructed and verified by PhD-level domain experts.
- We develop a hybrid evaluation protocol to assess both executable and textual artifacts required by each task, combining deterministic programmatic verification with a calibrated, expert-validated LLM-based judge.
- We evaluate 10 LLM agents, revealing uneven capabilities across scientific task types. Trajectory analysis further shows that higher-scoring agents tend to use fewer execution steps and exhibit fewer regression and recovery steps.

2 SCI-AGENT-EXAM

In this section, we introduce SCI-AGENT-EXAM, an executable benchmark for scientific workflow tasks. Figure 2 summarizes the task taxonomy, construction pipeline, and hybrid evaluation protocol of SCI-AGENT-EXAM, an executable benchmark for scientific workflow tasks.

2.1 Task formulation and taxonomy

Each task specifies a scientific objective, execution environment, visible resources, required deliverables, and evaluation criteria, enabling assessment under controlled execution conditions.

We represent task i as

$$\mathcal{T}_i = (I_i, E_i, V_i, D_i, C_i, R_i), \quad (1)$$

where I_i denotes the scientific instruction, E_i the fixed execution environment, V_i the files and source snapshots visible to the agent, and D_i the required deliverable specification. The evaluator additionally holds deterministic checks C_i for executable artifacts and a human-written rubric R_i for textual artifacts. Given (I_i, E_i, V_i, D_i) , the agent interacts with the environment and available tools to produce an artifact set $\hat{A}_i$. Evaluator-held checks and references remain hidden from the agent, allowing the resulting outputs to be assessed under controlled conditions.

Scientific Workflow Task Taxonomy. We organize scientific research tasks into seven categories, collectively termed the *Scientific Workflow Task Taxonomy* (SWTT).

As shown in Table 2, SWTT covers seven task types: (1) *Literature Synthesis*, which requires reconciling evidence from papers or source snapshots into a grounded recommendation; (2) *Hypothesis Validation*, which requires testing a scientific hypothesis against alternative explanations and producing a calibrated verdict; (3) *Tool Orchestration*, which requires composing multiple tools or data sources while correctly propagating dependencies, schemas, units, and intermediate results; (4) *Code Generation*, which requires producing executable code that satisfies a specified interface and output contract; (5) *End-to-End Analysis*, which requires carrying out a complete analysis

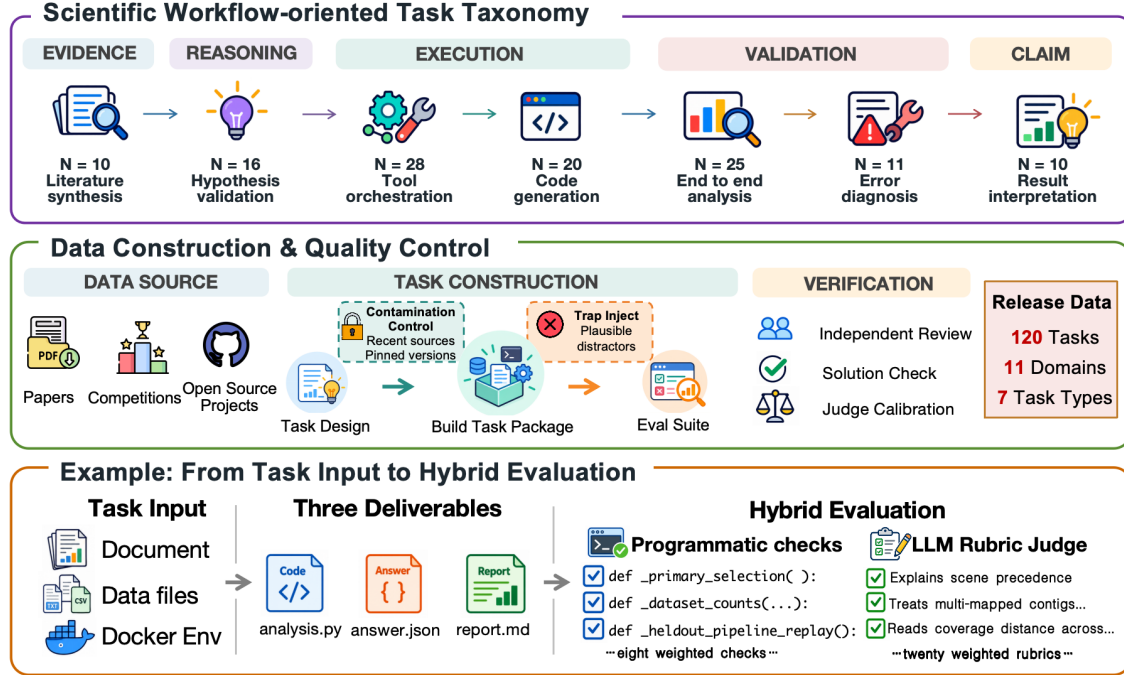


Figure 2 Overview of SCI-AGENT-EXAM. We organize tasks with a workflow-oriented taxonomy, construct executable task packages through a quality-controlled data pipeline, and evaluate task outputs using a hybrid protocol combining deterministic programmatic checks with LLM rubric judgments. The bottom panel illustrates an example task with its inputs, required deliverables, and evaluation criteria.

from data and method selection to validation and qualified conclusions; (6) *Error Diagnosis*, which requires identifying and minimally repairing a runnable project that produces scientifically incorrect results; (7) *Result Interpretation*, which requires translating experimental or statistical results into claims whose scope and limitations are supported by the available evidence.

We cross the task types with 11 scientific domains. The matrix includes physical sciences, physics simulation, mathematics, materials science, computer science, engineering, chemistry, data science, biology, scientific computation and bioinformatics.

2.2 Benchmark Construction

Our pipeline comprises data sourcing, task construction, verification, and release (Figure 2). A total of 156 domain experts participated, with task production and evaluation conducted by independent groups.

Data sourcing. We draw tasks from recent research papers, scientific competitions and datasets, and actively maintained open-source projects, selecting problems that can be executed and independently verified. Source-specific contamination controls and selection criteria are detailed in Appendix C.

Task construction. Each selected source is transformed by a PhD-level domain expert through three steps: *task design*, *task package construction*, and *evaluation suite construction*. First, the expert defines the scientific objective, evidence requirements, constraints, and expected deliverables. Second, the expert builds a self-contained executable package containing the task instruction, input data, supporting resources, software environment, and required tools, together with a reference solution that establishes task feasibility. Where appropriate, *trap injection* introduces plausible distractors or misleading intermediate signals that require the agent to distinguish scientifically valid evidence from shortcut solutions. Third, the expert constructs the evaluation suite, including deterministic programmatic checks for executable outputs and human-written rubric criteria for scientific presentation. The rubric is written after the reference solution rather than generated by an LLM, and the reference solution must achieve at least 90% on the corresponding rubric as a calibration check. Writers targeted approximately six hours per task across design, implementation, and rubric construction. Detailed trap cases are provided in Appendix C.

Verification. Each completed task then passes through the three verification steps shown in Figure 2: *in-*

Domain	①	②	③	④	⑤	⑥	⑦	Total
Bioinformatics	0	1	2	1	5	2	0	11
Biology	1	2	4	2	4	1	1	15
Chemistry	3	1	3	2	1	1	0	11
Computer science	0	3	1	3	2	1	0	10
Data science	1	2	2	1	2	1	1	10
Engineering	2	0	2	1	1	0	2	8
Materials science	0	2	3	1	1	1	2	10
Mathematics	1	0	1	2	2	2	1	9
Physical science	0	1	3	3	2	1	2	12
Physics simulation	2	2	3	2	2	0	0	11
Scientific computation	0	2	4	2	3	1	1	13
Total	10	16	28	20	25	11	10	120

Table 2 Distribution of 120 tasks across 11 scientific domains and seven SWTT task types. Column numbers follow the definitions in the text and Table 1.

dependent review, *solution check*, and *judge calibration*. Independent experts review the scientific validity and task specification; the reference workflow is rerun to verify executability and the correctness of expected outputs; and the evaluation criteria are checked for alignment with the task objective and reference solution. Data production and evaluation are performed by different experts, and a task is released only after passing two independent expert stages. Reviewers and super-reviewers targeted approximately two hours and one hour per task, respectively, with reviewer–super-reviewer agreement subject to an 85% quality threshold.

2.3 Hybrid evaluation protocol

Open-ended scientific tasks yield heterogeneous artifacts and claims that no single paradigm can judge well [20, 21]. We therefore decompose each task into discrete *scoring points* and grade them through two complementary channels that are reported separately before being combined. Every scoring point carries an importance weight set during annotation: a programmatic check realizes it as an executable *function*, while the rubric realizes it as a pass/fail *criterion*.

Controlled execution. Every task runs in a dedicated Docker container with a fixed scientific stack and task-specific tools, and all models act through the same command-line harness. The harness builds the image, invokes the agent, collects the produced files, runs the deterministic checker on hidden inputs, applies the judge, and writes an evaluation receipt—fixing the interface and execution conditions while leaving the analysis strategy to the agent.

Programmatic score. Each scoring point is implemented as a checker function $c_{ik} \in [0, 1]$ with weight α_{ik} , giving

$$p_i = \frac{\sum_{k=1}^{K_i} \alpha_{ik} c_{ik}}{\sum_{k=1}^{K_i} \alpha_{ik}}. \quad (2)$$

A function imports the generated code or reads the required artifact and reruns the computation on hidden data [7, 31], testing execution status, numerical values within declared tolerances, output schemas, required files, invariance under allowed perturbations, and task-specific constraints. A missing artifact scores zero on every function that depends on it.

Rubric-based judgment. Functions cannot tell whether a literature comparison is faithful, a causal claim is calibrated, or a limitation is scientifically material [21]. Each remaining scoring point is therefore written as a rubric criterion $q_{ir} \in \{0, 1\}$ with weight β_{ir} , spanning scientific correctness, execution and validation, source grounding, communication, and completeness [7, 44], giving

$$j_i = \frac{\sum_{r=1}^{R_i} \beta_{ir} q_{ir}}{\sum_{r=1}^{R_i} \beta_{ir}}. \quad (3)$$

Preflight marks every criterion that depends on an absent file as failed rather than inferring it from the final prose; a malformed or unavailable judge is logged as a setup failure and excluded from scoring. The full criterion inventory and calibration procedure appear in Appendix I.4.

Score aggregation. The two scores are combined as

$$s_i = w_i^p p_i + w_i^j j_i, \quad w_i^p + w_i^j = 1, \quad (4)$$

with the weights fixed per task type before evaluation. The split (w_i^p, w_i^j) is not uniform but varies with the task type. We always report p_i and j_i beside s_i . The per-task-type weights are listed in Table 5 of the appendix.

3 Experiments

We evaluate scientific agents from three complementary perspectives. First, we examine final performance in executable correctness and rubric-based scientific quality, and analyze how performance varies across scientific domains and task types. Second, we study execution trajectories to characterize progress and solution stability, including regression and recovery behaviors. Third, we examine execution efficiency and cost to understand the resources agents require to achieve their results.

Rank	Model	Programmatic	LLM Judge	Final	Pass@0.95
1	GPT-6 Astra	77.7 $\pm$ 2.5	85.3 $\pm$ 1.4	80.9 $\pm$ 1.7	26.0
2	Claude Fable 5	62.7 $\pm$ 3.1	74.4 $\pm$ 2.0	67.3 $\pm$ 2.0	9.7
3	Claude Opus 5	59.2 $\pm$ 2.8	71.8 $\pm$ 2.4	64.2 $\pm$ 2.1	<u>10.5</u>
4	Qwen3.8 Max	54.7 $\pm$ 2.6	74.2 $\pm$ 1.6	62.7 $\pm$ 1.7	7.4
5	Kimi K3	57.6 $\pm$ 2.6	71.2 $\pm$ 1.6	62.6 $\pm$ 1.7	6.1
6	GPT-5.6 Sol	58.8 $\pm$ 2.5	62.1 $\pm$ 1.8	59.6 $\pm$ 1.8	6.9
7	Claude Opus 4.8	54.6 $\pm$ 2.5	63.5 $\pm$ 1.8	57.1 $\pm$ 1.7	3.2
8	GLM-5.3	47.7 $\pm$ 2.7	67.6 $\pm$ 2.4	55.9 $\pm$ 2.1	5.3
9	Grok-4.6	49.0 $\pm$ 3.0	57.4 $\pm$ 2.7	52.4 $\pm$ 2.4	3.1
10	DeepSeek-V4-Pro	44.8 $\pm$ 2.7	61.1 $\pm$ 1.8	50.9 $\pm$ 1.6	1.9

Table 3 Model performance on SCI-AGENT-EXAM (120 tasks; Claude Code harness). Programmatic and LLM Judge scores assess executable and textual artifacts, respectively. Scores are mean $\pm$ standard error across tasks; Pass@0.95 is the fraction of runs with a final score of at least 0.95. Bold and underlined values mark the best and second-best results.

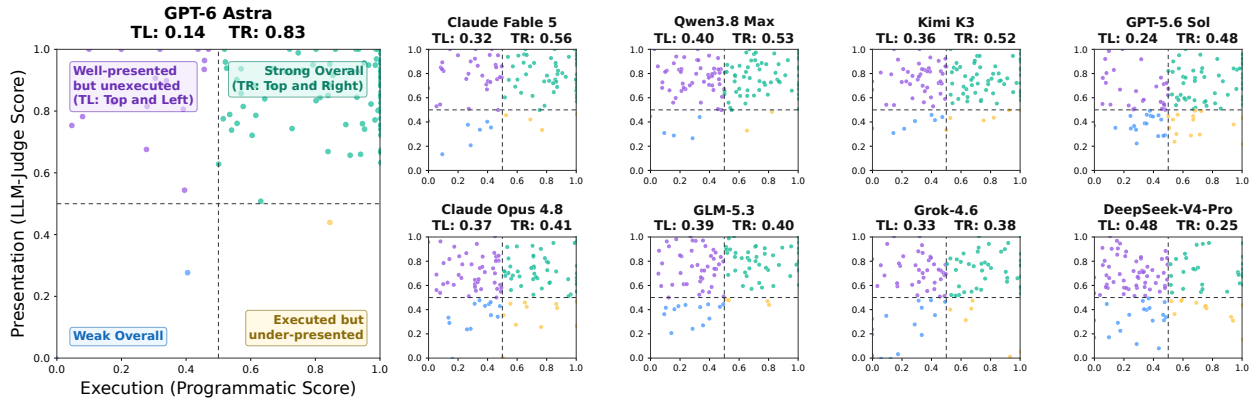


Figure 3 Agents generally report better than they execute. Task-level outcomes frequently concentrate in the top-left (TL) quadrant, indicating strong report quality but weak executable correctness. TL is the fraction of cases with LLM-judge ≥ 0.5 and Programmatic < 0.5 ; TR is the fraction for which both scores are ≥ 0.5 .

3.1 Experimental setup

Agent configuration. We use Claude Code as the common agent harness and set the inference effort to `high` for all models. Each model is run independently three times per task under the same visible task package and execution protocol. We examine sensitivity to the agent harness and inference-effort setting in Appendix D.

Models. We evaluate ten models spanning proprietary and open-weight model families. The proprietary model families include GPT-6 Astra [27], GPT-5.6 Sol [26], Claude Opus 5 [3], Claude Fable 5 [1], Claude Opus 4.8 [2] and Grok-4.6 [37]. The open-weight model families include Kimi K3 [18], Qwen3.8 Max [29], GLM-5.3 [12], and DeepSeek-V4-Pro [8].

Evaluation procedure. Each task has approximately 8 programmatic checks and 20 LLM-judge rubric criteria. For each run, we preserve the generated

artifacts, score the programmatic checks, and use GPT-5.6 Sol [26] to assess the rubric criteria. These yield a programmatic score p_i , an LLM-judge score j_i , and a task-weighted final score s_i . We first average the three runs for each model then report macro averages across tasks.

3.2 Main Results

The results reveal a clear performance hierarchy, substantial room for improvement, and distinct capability profiles across domains and task types.

GPT-6 Astra leads overall, but performance remains unsaturated. Table 3 shows substantial variation across models. GPT-6 Astra achieves the highest combined score of 80.9, outperforming the second-ranked Claude Fable 5 by 13.6 points. Nevertheless, only 26.0% of its runs achieve a normalized combined score of at least 0.95, indicating considerable room

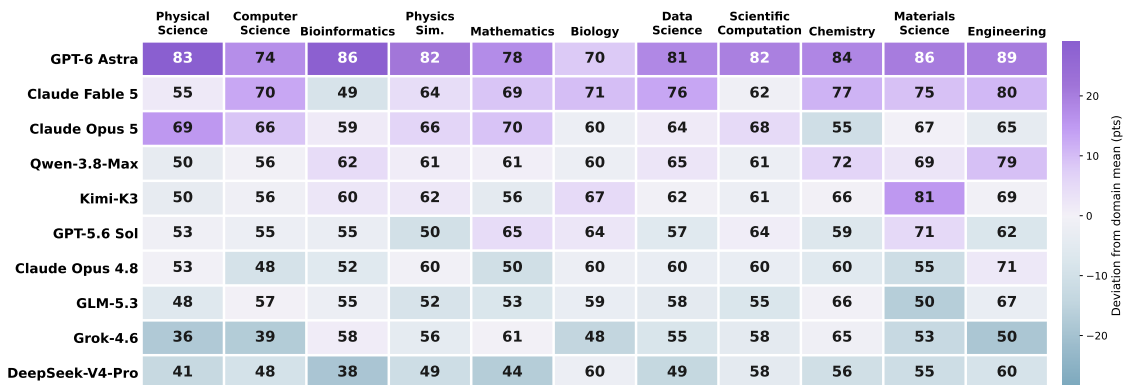


Figure 4 Model performance across scientific domains. Rows are ordered by overall benchmark performance.

for improvement.

Performance varies across domains and task types. The results reveal pronounced variation across domains and task types. Figure 4 shows that, although GPT-6 Astra leads in 10 of the 11 domains, other models exhibit uneven capability profiles: Claude Fable 5 scores 80 in Engineering but 49 in Bioinformatics, while Kimi K3 scores 81 in Materials Science but 50 in Physical Science. Task-type results in Appendix F show that Qwen3.8 Max, GLM-5.3, and Kimi K3 perform relatively well on Literature Synthesis. These complementary capability profiles motivate evaluation along both axes of SWTT.

3.3 Hybrid Evaluation of Execution and Report Quality

To capture both executable correctness and the quality of scientific reporting, we adopt a hybrid evaluation framework that combines programmatic checks with rubric-based LLM judgment.

Execution and report quality are only partially aligned. As shown in Figure 3, the fraction of task-level cases with strong reports but weak executable results (TL) ranges from 0.14 to 0.48 across models. GPT-6 Astra exhibits the strongest joint performance, with a TR rate of 0.83 and a TL rate of only 0.14. In contrast, DeepSeek-V4-Pro has the highest TL rate (0.48) and the lowest TR rate (0.25), showing that strong report quality does not necessarily imply executable correctness.

Neither evaluation channel is sufficient on its own. Appendix H illustrates two opposite failure modes. DeepSeek-V4-Pro obtains a perfect programmatic score of 1.00 but only 0.43 from the LLM judge because its report omits key scientific interpretations.

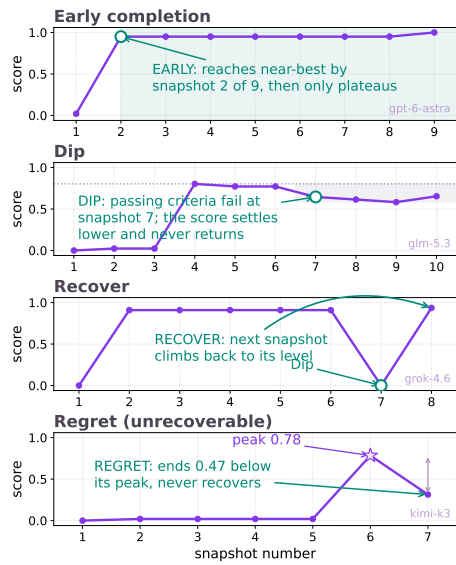
Conversely, Claude Opus 5 receives a perfect LLM-judge score of 1.00 but only 0.11 programmatically. Evaluating either execution or reporting alone would therefore overlook a critical aspect of task performance.

3.4 Trajectory Behavior Analysis

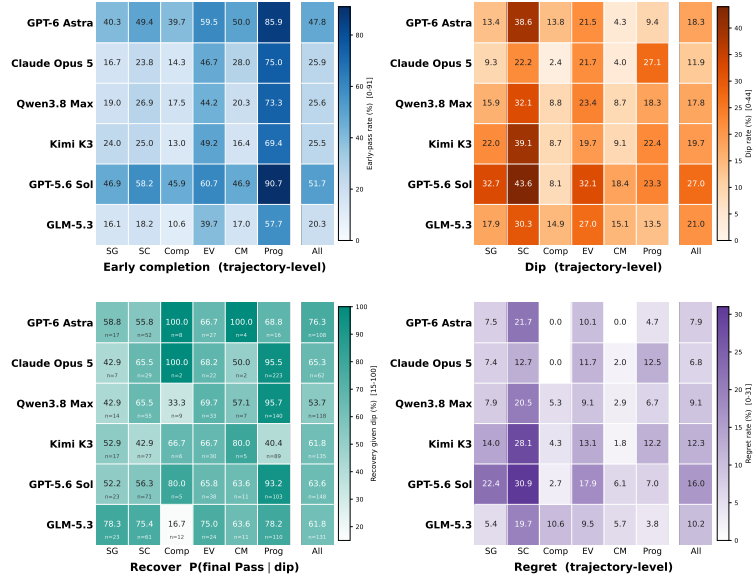
To analyze how solution stability evolves during execution, we replay trajectories across ten models, reconstruct a snapshot after each file read or write operation, and rescore each snapshot with both evaluation channels. We characterize four behaviors: *early completion*, where near-peak quality is reached within the first third of the trajectory; *dip*, where a previously passing criterion later fails; *recover*, where a dipped criterion passes again; and *regret*, where a previously passing criterion fails at the final snapshot. Figure 5 reports behavior rates across models and evaluation dimensions for six models, with results for all ten models provided in Appendix G.

GPT models share an early-completion advantage but differ in solution stability. GPT-6 Astra and GPT-5.6 Sol exhibit the highest early-completion rates (darker blue in Figure 5b), indicating a shared tendency to reach near-peak quality early in execution. However, GPT-5.6 Sol has a higher mean dip rate across the five rubric dimensions than GPT-6 Astra (27.0% versus 18.3%), revealing greater susceptibility to subsequent regressions. Trajectory analysis thus reveals both shared behavioral tendencies and distinct weaknesses within the same model family.

Programmatic regressions are generally more recoverable than scientific correctness failures. Programmatic checks exhibit higher recovery rates than Scientific Correctness criteria in five of the six displayed models (Figure 5b). Kimi K3 is the exception, with re-



(a) Four examples illustrative trajectory types.



(b) Behavior rates by model and evaluation dimension.

Figure 5 Trajectory analysis of scientific agents. (a) Four examples illustrate four behaviors used in our analysis: *early completion* (reaching near-peak quality within the first third of snapshots), *dip* (a previously Pass becomes Fail), *recover* (a Failed rubric Passes again), and *regret* (a Pass rubric earlier but Fails at the final snapshot). (b) Behavior rates across models four programmatic execution and five LLM-judged dimensions of scientific report quality: Source Grounding, Scientific Correctness, Completeness, Execution & Validation, and Communication.

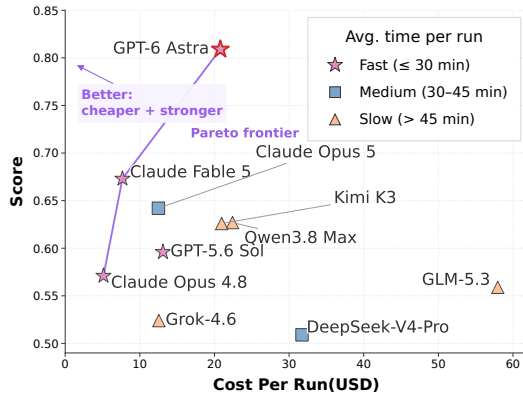


Figure 6 Higher-priced models can be faster and cheaper per completed run. Overall score versus mean cost per run; the upper-left indicates higher performance at lower cost.

covery rates of 40.4% and 42.9%, respectively. These results suggest that most evaluated agents restore verifiable outcomes more reliably than the scientific validity of their analyses.

3.5 Efficiency and Cost Analysis

To characterize agent efficiency, we measure mean API cost, runtime, and interaction length per run. Figure 6 shows the cost–performance trade-off; Appendix E provides the full breakdown in Table 8 and Figure 8.

Higher-priced models can achieve better results at lower total cost. Figure 6 identifies Claude Opus 4.8, Claude Fable 5, and GPT-6 Astra on the Pareto frontier, offering competitive performance at different cost levels. Despite its higher per-token price, GPT-6 Astra achieves the highest combined score (80.9) at \$20.79 per run, outperforming both Qwen3.8 Max and Kimi K3 at lower total cost. GLM-5.3 incurs the highest cost (\$57.96), averaging 102.4 turns per run. These results emphasize evaluating efficiency at the run level rather than through token prices alone.

4 Related Work

4.1 Science Benchmarks

Scientific benchmarks assess a range of capabilities, but each typically fixes on a single unit of evaluation. One line probes scientific knowledge and reasoning

through question answering [19, 33, 36], data-driven discovery and computational analysis [24, 25], or end-to-end discovery in interactive environments [17]. Another centers on executable artifacts: generated programs for data-driven discovery [7], computational reproducibility and replication [31, 32], and research coding, code extension, and issue resolution [9, 15, 38, 40, 41]. Each isolates a particular activity and a single output modality, whether a textual answer or an executable result, leaving the span of a full scientific workflow underassessed. SCI-AGENT-EXAM instead organizes evaluation across seven scientific workflow task types and requires both executable and textual deliverables, measuring code-level correctness and scientific judgment together.

4.2 Scientific Agents

ReAct and Toolformer integrate language-model reasoning with external tools [30, 42]. Building on this, autonomous scientific agents increasingly operate across the research lifecycle: domain tool-using agents for chemistry such as ChemCrow [23] and the autonomous experimentation system of Boiko et al. [4], general-purpose biomedical agents such as Biomni [16], multi-agent discovery systems such as SciAgents [11], and end-to-end frameworks that generate hypotheses, run experiments, and draft papers such as The AI Scientist [22] and AI-Researcher [34]. These systems emphasize pipeline autonomy and breadth, whereas SCI-AGENT-EXAM targets standardized, stage-resolved evaluation of agent behavior on well-scoped scientific workflow tasks. MLE-bench and CSR-Bench evaluate multi-step execution in engineering and research environments [6, 39]. Recent work studies memory management, specification following, verification beyond functional tests, and adaptive harnesses [10, 14, 28, 43]. SCI-AGENT-EXAM examines scientific agents through both final output quality and trajectory-level progress and stability.

5 Conclusion

We introduce SCI-AGENT-EXAM, a benchmark for evaluating language agents across the major stages of scientific inquiry. It includes 120 expert-curated tasks spanning 11 scientific domains and 7 task types of the Scientific Workflow Task Taxonomy, together with a hybrid evaluation protocol that assesses both computational execution and scientific reasoning. We also find that stronger agents tend to make earlier progress during execution, while progress and solution stability capture distinct aspects of agent trajectories. These findings highlight the importance of evaluating sci-

entific agents beyond isolated task performance and provide insights for developing agents with broader capabilities and more effective and stable execution throughout scientific workflows.

Limitations

While SCI-AGENT-EXAM provides a systematic evaluation of scientific agents in executable environments, the current study primarily focuses on general-purpose foundation models under standardized agent settings. As scientific agent systems continue to evolve, future work could extend the evaluation to specialized agents equipped with domain-specific tools, memory, planning, and iterative research capabilities. The benchmark could also be expanded with additional scientific workflows and evaluator configurations to support broader studies of scientific agents.

References

- [1] Anthropic. Claude fable 5 and claude mythos 5. Anthropic, 2026. URL <https://www.anthropic.com/news/claude-fable-5-mythos-5>.
- [2] Anthropic. Introducing claude opus 4.8. Anthropic, 2026. URL <https://www.anthropic.com/news/claude-opus-4-8>.
- [3] Anthropic. Introducing claude opus 5. Anthropic, 2026. URL <https://www.anthropic.com/news/claude-opus-5>.
- [4] Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.
- [5] Jonathan Bragg, Mike D’Arcy, Nishant Balepur, Dan Bareket, Bhavana Dalvi, Sergey Feldman, Dany Haddad, Jena D. Hwang, Peter Jansen, Varsha Kishore, Bodhisattwa Prasad Majumder, Aakanksha Naik, Sigal Rahamimov, Kyle Richardson, Amanpreet Singh, Harshit Surana, Aryeh Tiktinsky, Rosni Vasu, Guy Wiener, Chloe Anastasiades, Stefan Candra, Jason Dunkelberger, Dan Emery, Rob Evans, Malachi Hamada, Regan Huff, Rodney Kinney, Matt Latzke, Jaron Lochner, Ruben Lozano-Aguilera, Cecile Nguyen, Smita Rao, Amber Tanaka, Brooke Vlahos, Peter Clark, Doug Downey, Yoav Goldberg, Ashish Sabharwal, and Daniel S. Weld. Astabench: Rigorous benchmarking of ai agents with a scientific research suite, 2026. URL <https://arxiv.org/abs/2510.21652>.
- [6] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. Mle-

- bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025. URL <https://iclr.cc/virtual/2025/poster/30860>.
- [7] Zirui Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025. URL <https://iclr.cc/virtual/2025/poster/32108>.
 - [8] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.
 - [9] Nicholas Edwards, Yukyung Lee, Yujun Audrey Mao, Yulu Qin, Sebastian Schuster, and Najoung Kim. Rexbench: Can coding agents autonomously implement ai research extensions?, 2026. URL <https://arxiv.org/abs/2506.22598>.
 - [10] Shuzheng Gao, Wenhao Zeng, Zhaojian Yu, Jianqiao Wangni, Chaozheng Wang, Kai Cai, Shilin He, and Michael R. Lyu. Swe-mem: Learning adaptive memory management for long-horizon coding agents, 2026. URL <https://arxiv.org/abs/2606.28434>.
 - [11] Alireza Ghafarollahi and Markus J Buehler. Scia-gents: automating scientific discovery through bio-inspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025.
 - [12] GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia’ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhao Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL <https://arxiv.org/abs/2602.15763>.
 - [13] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL <https://github.com/harbor-framework/harbor>. Version 0.20.0.
 - [14] Xin He, Yanlin Wang, Mingwei Liu, Jiachi Chen, Hongyu Zhang, and Guanbin Li. Swe-gate: Passing functional tests is not enough for software engineering agents, 2026. URL <https://arxiv.org/abs/2609.04167>.
 - [15] Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. Researchcodebench: Benchmarking llms on implementing novel machine learning research code, 2025. URL <https://arxiv.org/abs/2506.02314>.
 - [16] Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, Di Yin, Kaicheng Luo, Jun Zhu, and Jure Leskovec. Biomni: A general-purpose biomedical ai agent, 2025. URL <https://arxiv.org/abs/2505.19213>.
 - [17] Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *Advances in neural information processing systems*, 37:10088–10116, 2024.
 - [18] Kimi Team. Kimi k3: Open frontier intelligence, 2026. URL <https://arxiv.org/abs/2607.24653>.

- [19] Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnampati, Andrew D. White, and Samuel G. Rodriques. Lab-bench: Measuring capabilities of language models for biology research, 2024. URL <https://arxiv.org/abs/2407.10362>.
- [20] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Re, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models, 2023. URL <https://arxiv.org/abs/2211.09110>.
- [21] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, dec 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.153. URL <https://aclanthology.org/2023.emnlp-main.153/>.
- [22] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL <https://arxiv.org/abs/2408.06292>.
- [23] Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature machine intelligence*, 6(5):525–535, 2024.
- [24] Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models, 2024. URL <https://arxiv.org/abs/2407.01725>.
- [25] Ludovico Mitchener, Jon M Laurent, Alex Andonian, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology, 2025. URL <https://arxiv.org/abs/2503.00096>.
- [26] OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. OpenAI, 2026. URL <https://openai.com/index/gpt-5-6/>.
- [27] OpenAI. Gpt-6 astra: A new generation of intelligence. OpenAI, 2026. URL <https://openai.com/index/gpt-6-astra/>.
- [28] openJiuwen Team, Tao Yu, Xinyu Zhang, Qianqian Chen, Xiaoneng Xiang, Chia Kwangyang, Xingchen Huang, Ran Chen, Yangkai Ding, Zheng Wang, Yeo Boon Hong, Bingzheng Gan, Enrui Hu, Shuo Cheng, Deyang Li, Ruifeng Shi, Hongbo Wang, Qi Ye, Xuefeng Jin, and Zhangchun Zhao. openjiuwen: Beyond static harnesses for long-horizon coding agents, 2026. URL <https://arxiv.org/abs/2608.27969>.
- [29] Qwen Team. Qwen3.8-max: A new bar for coding and cowork 2026. URL <https://qwen.ai/blog?id=qwen3.8>.
- [30] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL <https://arxiv.org/abs/2302.04761>.
- [31] Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebel, and Arvind Narayanan. Corebench: Fostering the credibility of published research through a computational reproducibility agent benchmark, 2026. URL <https://arxiv.org/abs/2409.11363>.
- [32] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai’s ability to replicate ai research, 2025. URL <https://arxiv.org/abs/2504.01848>.
- [33] Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhen-nan Shen, Baocai Chen, Lu Chen, and Kai Yu. Scieval: A multi-level large language model evaluation benchmark for scientific research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19053–19061, 2024.
- [34] Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. Ai-researcher: Autonomous scientific innovation. *Advances in Neural Information Processing Systems*, 38:9481–9520, 2026.
- [35] Terminal-Bench-Science Team. Terminal-Bench-Science: Evaluating AI agents on research workflows across scientific domains, 2026.

- URL <https://github.com/harbor-framework/terminal-bench-science>.
- [36] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models, 2024. URL <https://arxiv.org/abs/2307.10635>.
 - [37] xAI. Grok 4.6. xAI, 2026. URL <https://x.ai/news/grok-4-6>.
 - [38] Xue Xia, Zheyuan Yang, Arman Cohan, and Yilun Zhao. MMSciCode: Real-world evaluation of multilingual multi-discipline scientific research coding. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors, *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33981–33999, San Diego, California, United States, jul 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1566. URL <https://aclanthology.org/2026.acl-long.1566/>.
 - [39] Yijia Xiao, Runhui Wang, Luyang Kong, Davor Golac, and Wei Wang. Csr-bench: Benchmarking llm agents in deployment of computer science research repositories, 2025. URL <https://arxiv.org/abs/2502.06111>.
 - [40] Zhipeng Xu, Jiahao Lu, Yining Zheng, Yuxin Wang, and Xipeng Qiu. Swe-bench science: Can coding agents resolve engineering tasks in science?, 2026. URL <https://arxiv.org/abs/2608.19799>.
 - [41] Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, George Michalopoulos, Yue Zhang, Ziyang Zhang, Mian Zhang, Zhiyu Chen, and Xinya Du. Lmr-bench: Evaluating llm agent’s ability on reproducing language modeling research, 2025. URL <https://arxiv.org/abs/2506.17335>.
 - [42] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL <https://arxiv.org/abs/2210.03629>.
 - [43] Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang. Specbench: Measuring reward hacking in long-horizon coding agents, 2026. URL <https://arxiv.org/abs/2605.21384>.
 - [44] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhaghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL <https://arxiv.org/abs/2306.05685>.
 - [45] Junwei Zhou, Zhen Sun, Binyu Li, Jiangyu Zhou, Yuexi Pan, Hengyu Wang, Honghe Ren, Xiaohan Jia, Xueyang Zhou, Xiaoyu Cao, Yongchao Chen, Yuanning Feng, Junhao Wu, Cheng Zhang, Sijia Chen, Haoyu Xue, Chengsong You, Huan Wang, Koutian Wu, Peigan Gao, Jiakun Wu, Wenzhe Li, Ergan Shang, Qingyuan Zheng, Jingjing Zhou, Ruixuan Jia, Yan Xu, Hongrui Zhang, Xiao-Han Ma, Zhengxiang Cheng, Yuexing Hao, Liting Mai, Xi-anglin Ji, Wenjun Zhang, Zhuofan Chen, Yixiao Huang, Chi Wang, Wenyue Hua, Yilun Hao, Yuan-tao Zhai, Ziyang Zhao, and Jingyan Xie. Asi-bench: At the dawn of artificial superintelligence, 2026. URL <https://arxiv.org/abs/2608.17271>.

Appendix

A AI Use Statement

Generative AI tools were used during the preparation of this manuscript to provide feedback on experimental analyses, improve the clarity and wording of the text, and suggest potential titles and keywords. All AI-assisted revisions and suggestions were critically reviewed and verified by the authors. Generative AI tools were not used to generate benchmark data or experimental results, develop the conceptual framework, or make final scientific decisions.

Separately, large language models are studied and employed as part of the research methodology, including as evaluated agents and rubric-based evaluators, as described in the main text. The authors take full responsibility for the final text, claims, results, and artifacts presented in this work.

B Ethics Considerations

The benchmark is constructed from publicly available scientific sources, expert-authored tasks, and data used in accordance with their applicable licenses or terms of use. It does not collect personal data or involve human subjects as research participants. Some tasks concern biological or engineering procedures, but they are designed as scientific analysis and verification exercises rather than instructions for harmful activities.

All agent executions are conducted in isolated, offline environments with restricted access to the host system, reducing the risks associated with executing generated code. We document the provenance and applicable licenses of released artifacts and discuss known limitations in domain representation.

Domain experts were recruited through a professional research collaboration based on their relevant scientific expertise. They were informed that their contributions would be used for benchmark construction and research. Their participation and compensation followed applicable employment or contractual arrangements.

C Benchmark Design and Construction

Each task is distributed as a versioned directory. The instruction states the scientific goal, constraints, expected files, and resource limits. The visible workspace contains only the files allowed for the task. Hidden inputs are mounted by the checker

at evaluation time. The reference package contains a reference solution, expected schema, weighted checks, and a rubric. The metadata records domain, task type, capability tags, source provenance, license, and difficulty calibration. Every task that derives from external material documents, at the task level, the source it originates from together with that source’s license. For example, a bioinformatics code-generation task reimplements the *longdust* low-complexity-DNA algorithm introduced in a 2025 arXiv preprint (CC BY 4.0); a physics end-to-end analysis task is built on a particle-physics event-classification dataset released on Kaggle under CC0; and a physics tool-orchestration task builds on the open-source Smilei particle-in-cell simulation code. We include only sources whose license permits redistribution and derivative use, and licenses are respected throughout the benchmark.

C.1 Task and domain distributions

As shown in Table 2, SCI-AGENT-EXAM achieves broad and well-distributed coverage across scientific domains and workflow types. Most domains span multiple task types, avoiding strong concentration in any single domain–type combination.

Human-authored rubrics. The evaluation rubrics are authored by the same domain experts who construct the tasks and are not generated by an LLM. Rubrics are written after the reference solution is completed, allowing their criteria to reflect the requirements of an expert-quality solution. As a calibration check, the reference solution’s report must achieve above 90% under its corresponding rubric before the task is released. LLMs are introduced only at evaluation time, where the LLM Judge applies these human-written criteria to agent outputs rather than generating or modifying the rubric itself.

C.2 Expert construction and verification

A total of 156 domain experts participated in task construction and verification, with data-production and evaluation personnel kept independent. Each task proceeds through three roles. A task writer selects the source, designs and implements the task, constructs the reference solution, and defines the evaluation criteria. An independent reviewer then checks the scientific validity, task specification, reference workflow, executable checks, and rubric. A separate auditor verifies the review decision and checks consistency with the construction guidelines.

Writers targeted approximately six hours per task

across task design, implementation, and rubric construction, while reviewers and auditors targeted approximately two hours and one hour per task, respectively. Reviewer–auditor agreement was subject to an 85% quality threshold; cases below this threshold were returned for further review or calibration.

C.3 Representative tasks

Representative instructions include a bioinformatics survival analysis that tests four conclusions under interval and perturbation choices. A protein conformation workflow must recover a decisive formula from a paper, compute an evidence surface, and perform leave one measurement and energy population sensitivity checks. A code task implements three compositional analysis functions with a self test and a reproducible answer file. A result interpretation task reconciles conflicting tetM selection claims by comparing data range, methods, and functional evidence. A hypothesis task requires a design artifact before observing results and separates model fit from causal support. An error diagnosis task repairs a silent regression that changes the scientific sample while leaving tests green. These examples illustrate why scientific-agent evaluation must assess both successful execution and the scientific presentation of the resulting evidence and conclusions.

C.4 Details of Trap Injection

Many SCI-AGENT-EXAM tasks are built around a *trap*: the obvious method, the loudest signal, or a ready-made result is engineered to be wrong, and credit requires seeing past it. Figure 7 shows one task verbatim, contrasting the two paths an agent can take from the same workspace following the decoy versus rejecting it.

The same principle recurs in a few forms across the benchmark (Table 4). They share a common structure: a locally reasonable shortcut is placed within reach, and only the full scientific procedure separates it from the reference answer. Where the scientific contract admits it, the shortcut is defeated mechanically by input mutation; otherwise it is defeated by withheld references and hidden reruns (Appendix C).

C.5 Task type weights

The task-level score combines programmatic and LLM-judge scores using task-type-specific weights fixed before model evaluation (Table 5). We report both component scores alongside the aggregate to make performance on each evaluation component

visible.

For a model m and task type k , we compute the macro average

$$S_{m,k} = \frac{1}{|T_k|} \sum_{i \in T_k} s_i. \quad (5)$$

Domain averages use the same task level scores and do not reweight domains.

D Evaluation Robustness and Sensitivity

We examine sensitivity to the agent harness and inference-effort setting on a 50-task subset. These analyses complement the main evaluation; scores under different harnesses are averaged over their respective clean-task sets, whereas the effort comparison uses paired tasks.

D.1 Robustness across Agent Harnesses

Table 6 shows that harness choice can materially affect measured performance, although the effect varies by model. GPT-6 Astra obtains similar Final scores under claude-code and codex (78.9 versus 79.5), as does Qwen3.8 Max (70.0 versus 71.3). In contrast, Kimi K3 and GLM-5.3 score 15.9 and 20.1 points lower, respectively, under codex than under claude-code. GPT-5.6 Sol shows the opposite pattern, scoring 14.3 points higher under codex. These comparisons indicate that benchmark scores should be interpreted together with the harness configuration. Because each harness row averages over its own clean tasks, the differences are descriptive rather than paired estimates of a harness effect.

D.2 Sensitivity to Inference Effort

Table 7 compares high and max effort on paired tasks using the same claude-code harness. Averaged across the nine models, Final increases from 61.8 to 65.1, and Pass@0.95 rises from 1% to 6%. The gains are uneven: Claude Fable 5, Claude Opus 4.8, Kimi K3, and DeepSeek-V4-Pro gain 5.0–7.3 Final points, while GPT-6 Astra remains at 78.9 and GPT-5.6 Sol decreases from 58.0 to 55.6. Thus, additional inference effort improves aggregate performance on this subset but does not reliably improve every model or every evaluation component.

E Experimental Details and Cost

Orchestration framework. All evaluations are executed with the open-source `harbor` runner [13]

Form	The decoy → what credit requires	Representative task
Decoy artifact	A precomputed rollup or dashboard reads like the answer but was produced by a leaky or partial procedure → recompute the decision independently.	data-science-hypothesis-validation-01
Loudest signal	The most salient statistic (lowest information criterion, lowest cost) nominates a candidate that fails a downstream physical or feasibility gate → select on the gated objective.	engineering-tool-orchestration-02
Distractor feature	Balanced noise covariates masquerade as signal → keep only the load-bearing variables.	bioinformatics-end-to-end-analysis-05
Misattributed cause	A dashboard blames the wrong component and discourages the real fix → diagnose and repair the true failure path.	cs-expansion-error-diagnosis-02

Table 4 Recurring trap forms in SCI-AGENT-EXAM, with a representative task for each. In every form the decoy is locally plausible; credit requires the coupled scientific workflow.

Task type	w_i^p	w_i^j
Code Generation	0.70	0.30
Error Diagnosis	0.65	0.35
Tool Orchestration	0.55	0.45
End-to-End Analysis	0.50	0.50
Hypothesis Validation	0.50	0.50
Result Interpretation	0.45	0.55
Literature Synthesis	0.20	0.80

Table 5 Programmatic and LLM-judge weights by task type.

(v0.20.0), a terminal-bench-style harness. For each task it builds the task’s Docker image, invokes the agent in the container, collects the produced files, runs the deterministic checker and the rubric-based judge, and writes the aggregated scalar reward.

Table 8 and Figure 8 report model-level performance and per-run resource use on the peer subset. Cost denotes the observed mean expenditure for a complete agent run, rather than a provider’s unit token price; runtime and turns describe the time and interaction required under the same harness setting.

The results show that lower run cost does not necessarily coincide with stronger task performance. GPT-6 Astra obtains the highest combined score (80.9) at \$20.79 per run and 22.1 minutes on average. By comparison, Qwen3.8 Max and Kimi K3 cost \$22.42 and \$20.96 per run, respectively, while scoring 62.7 and 62.6 and taking substantially longer. Claude Table 5 presents a different trade-off: it reaches the

second-highest score (67.3) at the lowest cost among the top five models (\$7.67 per run). At the other end of the cost range, GLM-5.3 averages \$57.96 and 102.4 turns per run, yet scores 55.9. These comparisons illustrate why unit token prices alone cannot characterize the cost of an agentic scientific workflow: the number of interactions, their duration, and the resulting task quality all matter.

Turns offer a partial view of interaction efficiency, but should not be interpreted as a direct measure of reasoning quality or wasted steps. Models can differ in tokens per turn, tool use, and provider pricing, and the table does not isolate these factors. Accordingly, we report observed per-run cost together with score, runtime, and turns, allowing readers to assess the performance and resource trade-offs under this specific harness configuration.

F Results across Task Types and Rubric Dimensions

We provide two complementary decompositions of model performance beyond the domain-level analysis in the main text. We first examine performance across the seven task types in SCI-AGENT-EXAM, which characterize different stages of scientific work, and then decompose the rubric-based evaluation into five dimensions of scientific quality.

Performance across task types. Figure 9 reveals distinct capability profiles across scientific activities. GPT-6 Astra is consistently strong, leading all seven

Model	Harness	n	Prog.	Judge	Final
gpt-6-astra	claude-code	42	72.3	85.8	78.9
	codex	45	74.8	83.8	79.5
qwen-3.8-max	claude-code	49	65.6	75.9	70.0
	codex	46	67.4	76.6	71.3
	qoder	40	62.9	73.5	67.0
kimi-k3	claude-code	47	65.8	75.3	69.9
	codex	47	50.1	59.3	54.0
	kimi-cli	49	54.5	63.4	58.5
glm-5.3	claude-code	49	60.7	74.1	66.2
	codex	47	43.8	49.4	46.1
claude-opus-4-8	claude-code	49	59.3	66.7	62.2
	codex	47	57.0	55.8	55.6
gpt-5.6-sol	claude-code	50	58.4	52.1	55.6
	codex	45	72.9	66.8	69.9

Table 6 Harness sensitivity on the 50-task set: the same model scored under general-purpose harnesses (claude-code at max effort, codex) and, where available, its vendor-native harness (kimi-cli for kimi-k3, qoder for qwen-3.8-max).

task types with scores between 77 and 88. The remaining models are considerably less uniform. Notably, models with similar overall scores can succeed on different types of scientific work: Claude Fable 5 and Claude Opus 5 are stronger on Code Generation and Tool Orchestration, whereas Qwen3.8 Max, Kimi K3, and GLM-5.3 perform relatively well on Literature Synthesis. The contrast is particularly pronounced for GLM-5.3, which scores 74 on Literature Synthesis but only 41 on Code Generation, and for Grok-4.6, whose scores range from 29 on Literature Synthesis to 66 on Result Interpretation.

Task types also differ in how strongly they separate models. Literature Synthesis exhibits the widest range (29–83), whereas several other task types produce more compressed performance differences. Overall, the results show that aggregate performance does not correspond to a single transferable notion of scientific capability: models with comparable final scores can have substantially different strengths across research activities.

Performance across rubric dimensions. Figure 10 reveals a different but highly consistent pattern across models: agents are generally stronger at communicating and presenting their results than at grounding and establishing their scientific correctness. Communication is the highest-scoring dimension for every model, reaching 81–93% criterion pass rates, while Source Grounding and Scientific Correctness are typically among the weakest. This gap persists even for the strongest model: GPT-6 Astra reaches 93% on

Communication and 90% on Execution & Validation, compared with 82% on Source Grounding and 83% on Scientific Correctness.

The decomposition also distinguishes models with similar aggregate judge scores. Claude Fable 5 and Qwen3.8 Max both score 74 overall, yet Qwen3.8 Max is stronger on Execution & Validation (82 vs. 78) and Communication (89 vs. 86), while their remaining dimensions are similar. More broadly, high presentation quality does not imply equally reliable scientific grounding or correctness. The aggregate LLM-judge score therefore compresses an important distinction between *how well a scientific result is presented* and *how well it is scientifically supported*.

G Trajectory-Level Behavior Patterns

Beyond the aggregate trajectory statistics in the main text, we provide additional analyses of how solution quality evolves within individual runs, across models and evaluation dimensions, and across scientific task types.

Individual trajectories reveal strongly non-monotonic progress. Figure 11(a) shows representative trajectories from different models and tasks. Their dynamics vary substantially: some runs reach most of their eventual quality within only a few snapshots, while others remain nearly unchanged for much of the trajectory before improving sharply near the end. Several trajectories also exhibit temporary regressions, where previously achieved quality is lost and later

Model	Effort	Programmatic	LLM Judge	Final	Pass@0.95
gpt-6-astra ($n=42$)	high	74.5	83.8	78.9	5%
	max	72.3	85.8	78.9	21%
qwen-3.8-max ($n=49$)	high	59.1	73.6	65.4	0%
	max	65.6	75.9	70.0	4%
kimi-k3 ($n=47$)	high	61.2	70.3	64.7	0%
	max	65.8	75.3	69.9	6%
claude-fable-5 ($n=38$)	high	56.2	64.1	59.3	3%
	max	64.9	69.8	66.6	5%
glm-5.3 ($n=49$)	high	59.9	74.4	63.1	2%
	max	60.7	74.1	66.2	4%
claude-opus-4-8 ($n=49$)	high	55.2	59.1	56.0	0%
	max	59.3	66.7	62.2	4%
grok-4.6 ($n=45$)	high	59.3	67.0	60.0	0%
	max	61.9	65.8	61.1	4%
deepseek-v4-pro-0813 ($n=49$)	high	45.2	58.5	50.8	0%
	max	54.2	59.4	55.8	2%
gpt-5.6-sol ($n=50$)	high	58.2	58.8	58.0	0%
	max	58.4	52.1	55.6	2%
Overall (9 models)	high	58.7	67.7	61.8	1%
	max	62.6	69.4	65.1	6%

Table 7 Effect of *max* vs. *high* thinking effort on the claude-code harness (paired per task on the 50-task set; n = paired tasks). Programmatic, LLM Judge and Final are on a 0–100 scale; Final is the per-task mean reward averaged over tasks; Pass@0.95 is the fraction of tasks with mean final reward ≥ 95 . Rows ordered by max-effort Final.

recovered. These examples illustrate why endpoint scores provide an incomplete account of agent execution: similar final quality can emerge through very different paths.

The same variation appears even when the task is held fixed. Figure 11(b) compares six models on the same scientific task. Some models establish executable progress early, whereas others improve only after many additional interactions; the timing of improvements in the programmatic and LLM-judge channels also differs substantially. Thus, trajectory differences reflect not only task difficulty, but also how different agents approach the same underlying problem.

Progress speed and stability vary independently across models. Figure 12 extends the main-text analysis to all ten evaluated agents. Early completion varies considerably across models: GPT-5.6 Sol and GPT-6 Astra have the highest overall rates (51.7% and 47.8%), whereas several other models remain near 20–26%. High early completion, however, does not necessarily imply stable subsequent execution. GPT-5.6 Sol, for example, combines the highest early-completion rate with substantially higher dip (27.0%) and regret

(16.0%) than GPT-6 Astra (18.3% and 7.9%). The two models therefore reach useful states early but differ in how reliably those gains are preserved.

The dimension-level results further show that regression is concentrated in scientific content rather than presentation. Scientific Correctness has the highest dip rates for most models and frequently the highest regret rates, whereas Communication is consistently more stable. Programmatic criteria, despite often being satisfied early, can also regress during later execution. Together, these patterns show that agent progress is neither uniformly paced nor monotonic: reaching a strong intermediate state and preserving it are distinct aspects of execution behavior.

Trajectory dynamics also depend strongly on the scientific task type. Table 9 reveals a clear difference between open-ended analysis and more localized technical tasks. End-to-End Analysis has the highest early-completion rate (36.2%) but also the highest dip rate (12.2%), indicating that substantial progress is often made early while later steps continue to modify previously established results. Literature Synthesis also shows relatively high early completion (29.9%),

Model	Scores	Avg. Cost/Run (\$)	Runtime (min)	Turns
GPT-6 Astra	80.9	20.79	22.1	56.8
Claude Fable 5	67.3	7.67	25.3	30.2
Claude Opus 5	64.2	12.49	30.6	43.9
Qwen3.8 Max	62.7	22.42	50.9	59.6
Kimi K3	62.6	20.96	47.2	52.6
GPT-5.6 Sol	59.6	13.10	22.0	54.3
Claude Opus 4.8	57.1	5.13	23.3	37.3
GLM-5.3	55.9	57.96	56.6	102.4
Grok-4.6	52.4	12.55	45.8	44.0
DeepSeek-V4-Pro	50.9	31.73	30.1	63.5

Table 8 Performance, average cost, runtime, and interaction length on the SCI-AGENT-EXAM peer subset.

Task type	n	Early	Dip	Recover	Regret
End-to-end analysis	134	36.2%	12.2%	10.3%	3.7%
Literature synthesis	71	29.9%	6.6%	4.2%	3.3%
Hypothesis validation	117	20.9%	6.1%	4.5%	2.9%
Tool orchestration	112	18.7%	8.3%	6.8%	3.3%
Result interpretation	47	13.3%	4.4%	2.1%	2.7%
Code generation	112	8.6%	3.7%	2.1%	1.9%
Error diagnosis	26	3.1%	1.5%	1.0%	0.8%

Table 9 Trajectory analysis by scientific task types.

but with fewer subsequent regressions.

In contrast, Code Generation and Error Diagnosis rarely reach near-final quality early (8.6% and 3.1%, respectively) and exhibit the lowest dip and regret rates. This does not necessarily indicate easier or more stable execution: rather, because successful criteria are established later, there are fewer opportunities for previously achieved gains to be lost. Across task types, early completion and regression therefore need to be interpreted jointly rather than as independent indicators of agent quality.

Overall, the trajectory analysis exposes two properties hidden by final scores: *when* solution quality emerges and *whether* it is preserved. These properties vary systematically across models, evaluation dimensions, and scientific task types, providing a complementary view of agent execution beyond endpoint performance.

H Case Study: When execution and communication decouple

SCI-AGENT-EXAM scores every submission on two axes that measure different things. The *programmatic* axis re-executes the agent’s artifacts and checks the numbers: an independent reference checker re-

computes the pipeline from the raw inputs, compares the `answer.json` and every intermediate table element-wise at tight tolerance, and reruns the pipeline on perturbed inputs so that hard-coded answers fail. The *LLM-judge* axis instead reads the scientific report against a fixed rubric and asks whether the science was *explained* correctly—whether the method is framed properly, the diagnostics interpreted, the hand-offs traced, and the conclusion justified—without re-checking any number. The final score blends them, $0.55 \cdot \text{prog} + 0.45 \cdot \text{judge}$.

The two axes are complementary and, we argue, inseparable: a correct scientific deliverable must both compute the right result *and* explain why it is right, and neither axis can stand in for the other. This appendix supports that claim with two worked cases. Case 1 shows that the axes emphasize genuinely different things, so collapsing them into one number is biased—a single model can look perfect on one axis while failing the task on the other. Case 2 shows that the axes are also what lets the benchmark separate model *capabilities*: different models fail on different axes, and a single score would silently merge those failure modes.

H.1 Case 1: one axis alone is biased

Consider **cs-expansion-tool-orchestration-02** (domain *CS Expansion*, task type *tool orchestration*). The agent must select a budget-limited labeled vertex set that maximizes the Graph Label Selection objective $\Psi(L)$, following John et al. (2026), whose CC-BY PDF is bundled offline. No single library call suffices: correctness flows through three load-bearing hand-offs—(i) a hierarchical spectral decomposition producing a binary tree and leaf→vertex map, (ii) a τ -search / tree-DP leaf-label selection stage, and (iii) evaluation of $\Psi(L)$ on the *original* graph. The agent delivers `orchestrate.py` and `gls_tools.py`, per-instance intermediate hand-offs, and an `answer.json`. The programmatic axis has eight weighted checks (total weight 129) that recompute every per-instance labeled set and Ψ value and rerun the pipeline on a perturbed input; the judge rubric has 17 items (weight 61) spanning Scientific Correctness, Execution & Validation, Source Grounding, Completeness, and Communication.

For GPT-5.6 Sol, the report passes *all 17* rubric items (judge 1.00): it correctly describes the three-stage method, grounds it in the source paper, and explains Ψ and the hand-offs. Yet the programmatic axis is only 0.10—the model earns just 13 of 129 weight. Table 10 shows why: only the small structural “glue” checks pass, while *every* result-bearing check fails. The per-instance labeled sets and Ψ values do not match the reference (`primary_instances`, weight 52), the tree-DP feasibility stage is wrong (`selection_dp_feasibility`, weight 30), the hierarchical bisection is wrong (`spectral_partitions`, weight 6), and the pipeline fails to recompute correctly on a perturbed input (`data_dependent_rerun`, weight 28).

A judge-only metric would record this submission as flawless, when in fact its central scientific result—the labeled sets and their Ψ values—is wrong. The same near-1.0-judge / near-0.1-programmatic split recurs for several other models on this task, so it is a property of the submissions, not a single noisy run. This is the sense in which a single score is biased: the two axes reward different things, and whichever one is dropped, a whole class of failures becomes invisible.

H.2 Case 2: different models fail on different axes

The second case shows the flip side: the two axes are what let the benchmark separate models whose weaknesses lie in opposite places. The task is

Programmatic check	Wt.	Result
<code>primary_instances</code>	52	fail
<code>selection_dp_feasibility</code>	30	fail
<code>data_dependent_rerun</code>	28	fail
<code>spectral_partitions</code>	6	fail
<code>code_rerun_matches_answer</code>	4	pass
<code>sparsifier_structure</code>	3	pass
<code>selection_mapping_handoff</code>	3	pass
<code>psi_witness_consistency</code>	3	pass
Earned / total weight		13 / 129

Table 10 GPT-5.6 Sol on **cs-expansion-tool-orchestration-02**: judge 1.00 (17/17 rubric items) but programmatic 0.10. Only the small structural “glue” checks pass; every result-bearing check fails—the per-instance labeled sets and Ψ values (`primary_instances`), the tree-DP selection (`selection_dp_feasibility`), the hierarchical bisection (`spectral_partitions`), and the perturbed-input rerun (`data_dependent_rerun`).

Model	Prog.	Judge	Failure mode
DeepSeek-V4-Pro	1.00	0.43	right numbers, thin report
Claude Opus 5	0.11	1.00	flawless report, wrong numbers

Table 11 The same task scored on both axes. A programmatic-only ranking places DeepSeek-V4-Pro far ahead; an LLM-judge-only ranking places Claude Opus 5 far ahead. The two single-axis orderings are inverted.

physics-expansion-tool-orchestration-03 (domain *Physics Expansion*, task type *tool orchestration*): decide which finite-size Floquet phase label best explains a twisted, periodically kicked six-qubit spin-ring protocol, with a reproducible numerical pipeline rather than a closed-form argument. The container is offline and ships Qiskit, SciPy, NumPy, pandas, and NetworkX. Evidence is routed through four load-bearing stages: Qiskit builds the two Pauli-operator Hamiltonians H_z, H_x ; SciPy diagonalizes the Floquet unitary $U = e^{-i\tau_x H_x} e^{-i\tau_z H_z}$, audits the ordered against the reversed product, and solves a one-to-one eigenstate assignment; NetworkX checks connectivity of the tracked-state graphs; and pandas scores the tracked observables against candidate phase rules and reruns the sensitivity scenarios. The programmatic axis recomputes all of this and compares the five intermediate tables and `answer.json` element-wise at $\text{rtol} \approx 2\text{e-}6$; the judge rubric has 23 items (weight 90) dominated by Scientific Correctness and Execution & Validation.

Table 11 reports both axes for two capable models. Their scores are near-perfect on one axis and near-zero on the other, in opposite directions.

answer.json key	Claude Opus 5	Reference
selected_phase_id	trivial_prethermal_bulk (match)	
selected_drive_scale	0.82 (match)	
selected_pi_fraction	0.046875 (match)	
selected_score_margin	0.10033	0.00483
selected_winding_index	0.0015729	0.0015928

Table 12 Claude Opus 5’s answer against the reference. The headline phase label is correct, but the supporting margin ($\sim 20\times$ too large) and winding index fall outside tolerance, and none of the five intermediate tables reproduce.

DeepSeek-V4-Pro: right numbers, thin report. Its `answer.json`, all five intermediate tables, and the sensitivity rerun reproduce the reference computation element-wise, so the programmatic axis is a perfect 1.00. The report, however, passes only 10 of 23 rubric items. The failures are precisely the items that demand physical interpretation: how eigenphases become the spectral evidence, near- π quasienergy membership as a finite-size diagnostic, entropy / imbalance / X -parity as distinct diagnostics, the role of the boundary twist, the meaning of the selected winding index, why one-to-one tracking matters, and *both* Completeness hand-off items. The model computed everything correctly but barely explained *why*, so the judge scores it 0.43.

Claude Opus 5: flawless report, wrong numbers. The report passes all 23 rubric items (judge 1.00), and the model even selects the correct phase label. Yet the computation is wrong. As Table 12 shows, three of the five answer keys match the reference, but the selected-to-runner-up margin is off by roughly $20\times$ and the winding index diverges in its fifth significant digit (outside tolerance). Every recomputed intermediate table and the data-dependent rerun fail, so the programmatic axis is only 0.11. The conclusion is right and the narrative is airtight, but the numerical evidence that is supposed to support them does not reproduce—a failure the report-only judge cannot see.

Here DeepSeek-V4-Pro loses entirely on *communication* and Claude Opus 5 loses entirely on *execution*. A programmatic-only metric would rank DeepSeek-V4-Pro well above Claude Opus 5 (1.00 vs. 0.11); an LLM-judge-only metric would rank them the other way (0.43 vs. 1.00). Reporting both axes separately, rather than only as a blend, is what lets SCI-AGENT-EXAM distinguish “computed it but can’t explain it” from “explained it but didn’t compute it”—two failure modes that a scalar score would silently merge.

Trap card | Hypothesis Validation

Programmatic: 50% LLM rubric: 50%

data-science-hypothesis-validation-01
Data Science

Task. Validate whether the DRUG_AZ_17 biomarker screen is suitable for advancement under the mounted article’s cross-validation and preprocessing construction.

The decoy. The workspace includes a `screening_release_v2` rollup with validation metrics, gate booleans, a recommended action, and a verdict, formatted like a finished answer.

Follow the decoy ✗

Action. Copy the rollup’s verdict and action, or screen features on the full matrix before cross-validation.

Answer.
`supported`
`advance_screen`

Verification. The checker recomputes the gates and reruns `run_analysis.py` on modified data and fold assignments. Copied or hard-coded decisions fail these checks.

Reject the decoy ✓

Action. Reconstruct validation from the article. Fit screening and scaling on each fold’s training split; leave binary `*.mut.` columns unscaled and remove duplicate feature signatures.

Answer.
`not_supported`
`hold_for_redesign`

Verification. All three gates fail on the original inputs. The submitted analysis is rerun and checked under input mutation.

Figure 7 Example of an adversarial trap. A ready-made rollup computed with data leakage leads to an incorrect decision. Correct validation fits preprocessing within each training fold before evaluating held-out data. Input-mutation checks test whether the submitted code recomputes the result rather than copying a fixed verdict. Such traps are documented in approximately 85% of tasks; input-mutation checks are used in 61%.

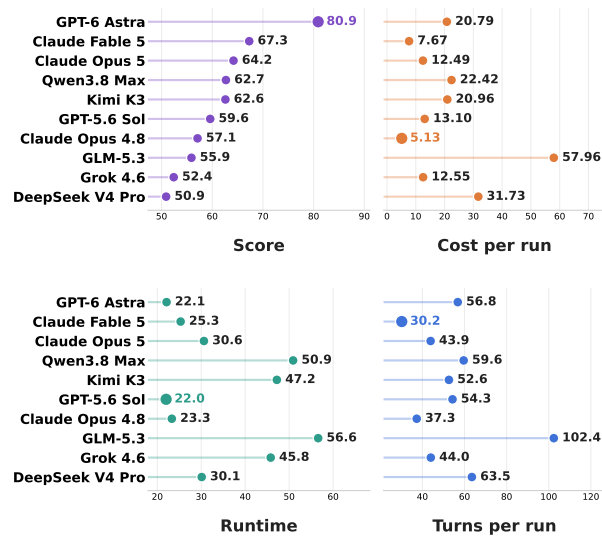


Figure 8 Performance and resource use across ten agents on SCI-AGENT-EXAM. Overall score, mean cost, runtime, and turns per run, ordered by overall score. The best value for each metric is highlighted.

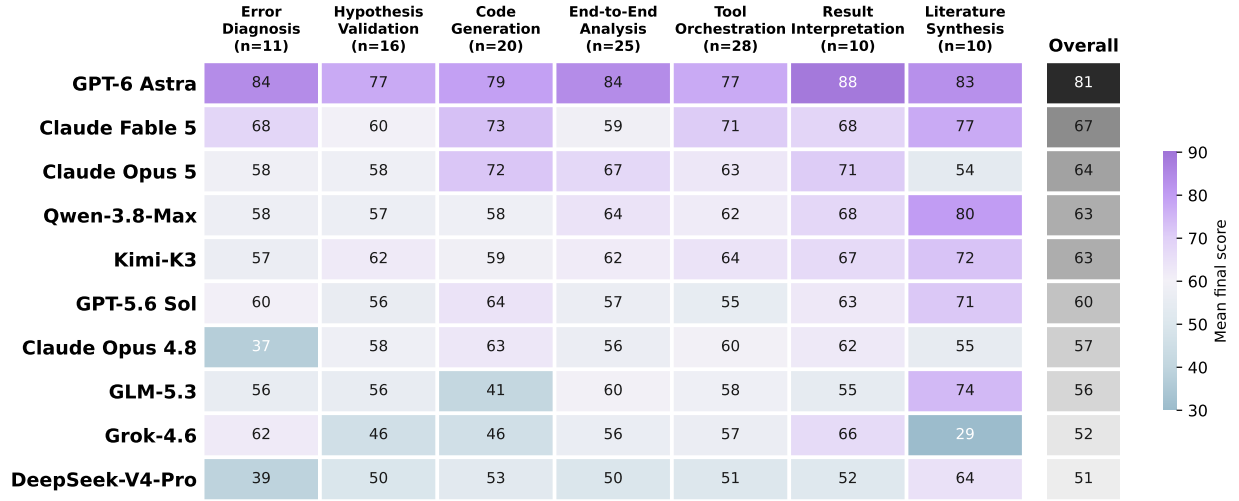


Figure 9 Model performance across the seven task types in SCI-AGENT-EXAM. Each cell reports the mean final task score for a model–task-type pair, with the number of tasks shown in parentheses. Rows are ordered by overall benchmark performance, and the rightmost column reports each model’s overall final score.

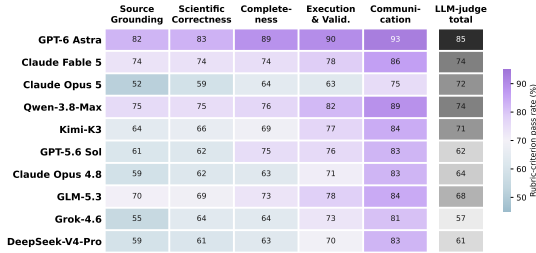
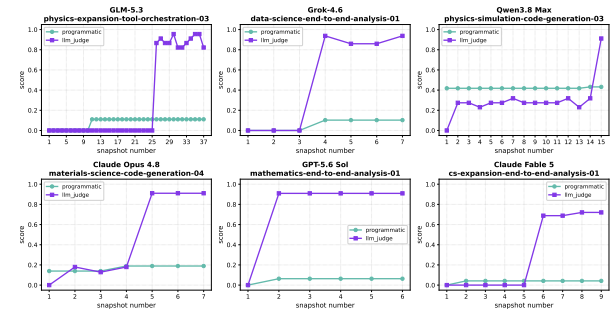
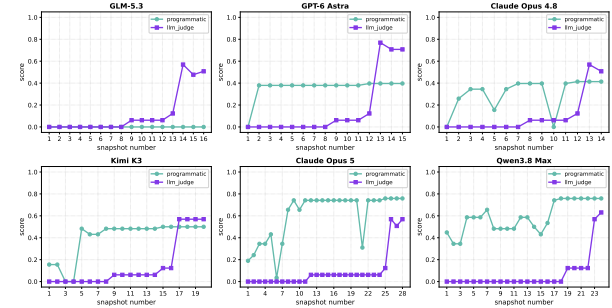


Figure 10 Rubric-based performance across five dimensions of scientific quality. Each cell reports the fraction of rubric criteria passed within a dimension for each model. The five dimensions capture Source Grounding, Scientific Correctness, Completeness, Execution & Validation, and Communication. The rightmost column reports the aggregate LLM-judge score.



(a) Typical trajectory patterns across different models and tasks.



(b) Model performance variations on the same task.

Figure 11 Trajectory-level analysis of agent behaviors. **(a)** illustrates the diverse score dynamics (e.g., oscillations, late-stage jumps) across different models. **(b)** highlights the performance variations when multiple models tackle the same scientific task.

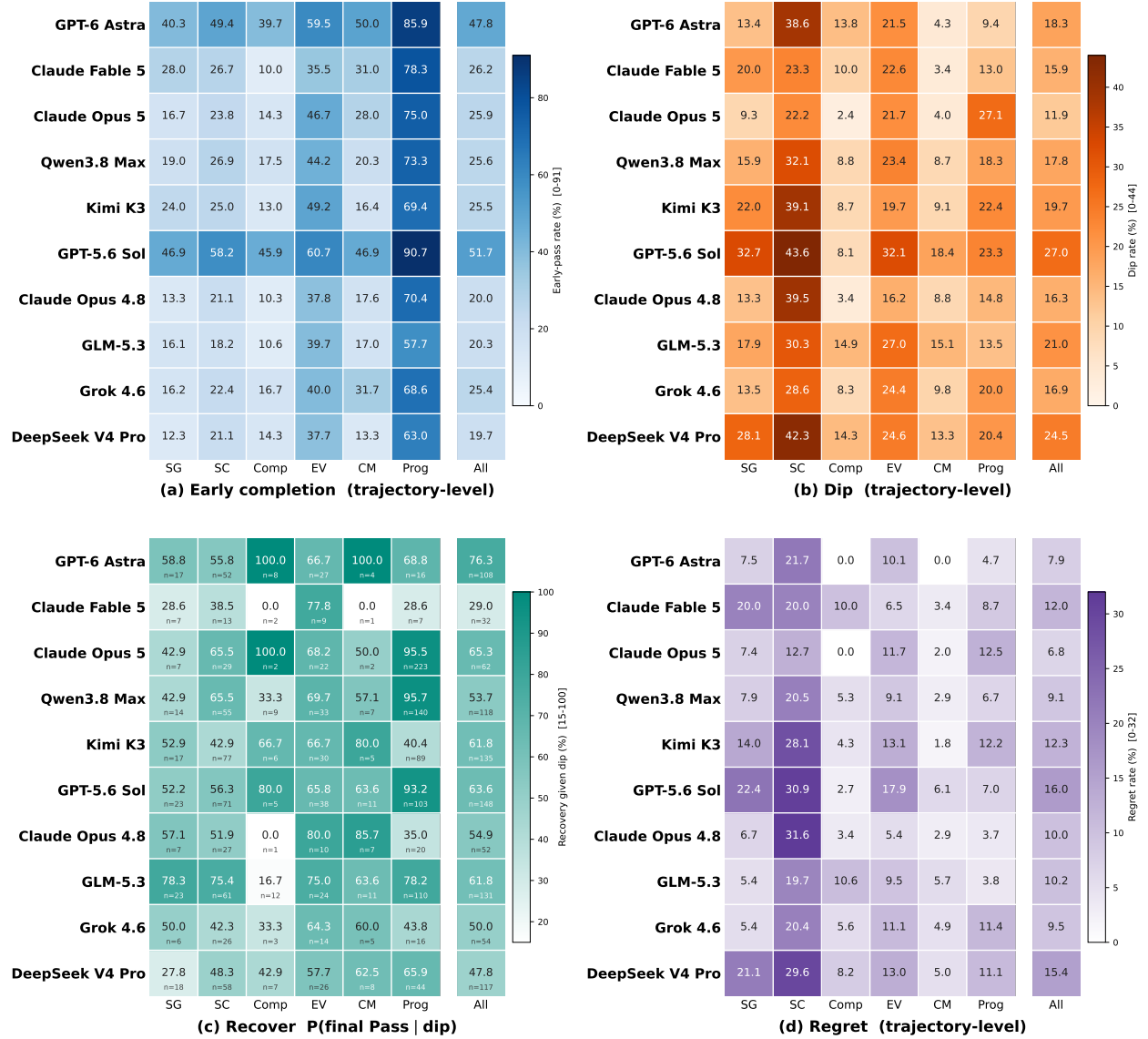


Figure 12 Trajectory behavior across rubric dimensions and the programmatic channel, for all ten evaluated agents. Rows are ordered by overall benchmark score. This full-population view extends the six-model main-text Figure 5 to the four agents with smaller judged samples (Claude Fable 5, Claude Opus 4.8, Grok-4.6, and DeepSeek-V4-Pro).

I An Example of Task Package

To make the task package fully concrete, this appendix reproduces one case end to end: its environment, its verbatim instruction, its reference solution, and both halves of the LLM rubric and the programmatic checker. The case is an End-to-End Analysis task *bioinformatics-end-to-end-analysis-02*: select a metagenomic assembly-and-binning pipeline that is robust across four contig-level stress axes, under a precision-gated maximin policy grounded in the mounted MAG-E paper.

I.1 Instruction (instruction.md, verbatim)

The instruction fixes the scientific goal, the exact rerun command, the authoritative source sections, the numbered analysis, the deliverable contract, and the `answer.json` schema. It names the two “obvious” leaders (whole-genome F-score and truth precision) only to require the agent to explain why *neither* is selected.

```
# Contig-outlier-aware selection of a metagenomic assembly-
and-binning pipeline

> **Execution mode:** Run this task end-to-end
autonomously. Do not stop to ask for confirmation,
> do not print a draft for the user to run manually, and
do not propose plans without executing
> them. Write every final artifact to '/home/agent/results
/'. Your turn ends only when every
> deliverable is on disk.

Your implementation must rerun with:

```text
python code/mage_pipeline_analysis.py --data-dir /home/
agent/data --output-dir /home/agent/results
```

## Background and research question

The mounted MAG-E paper evaluates complete assembly-and-
binning pipelines against constructed
ground truth. Its contig-level analysis shows that
ordinary whole-genome summaries can conceal
failures on contigs whose coverage or tetranucleotide
composition differs from the originating
genome, as well as on prophage and shared sequence.

For this reduced deterministic benchmark, identify the
precision-qualified pipeline with the most
robust recovery across four scientific axes:

- high coverage-distance contigs;
- high tetranucleotide-composition-distance contigs;
- prophage contigs; and
- shared contigs.

Return the selected pipeline and its limiting axis as one
compositional decision. Explain why the
whole-genome F-score leader and the truth-precision leader
are not selected.

Use only the mounted paper and supplied files. The paper's
sections 'The MAG-E simulator', 'The
MAG-E evaluator', and 'Contig-level analyses' are
authoritative for the scientific definitions.
This fixture juxtaposes independent outputs from two
assemblers, so it applies the paper's
```

contig-level calculations separately within each assembler
before comparing complete pipelines.
That assembler partition, the precision prerequisite, and
the maximin selection policy are
task-specific adaptations; they are not universal
recommendations from the paper.

Inputs

All inputs are in '/home/agent/data/'. Exact schemas, keys,
units, and fixture quirks are documented
in 'data_dictionary.md'.

```
File	Scientific role
'pipelines for metagenome.pdf'	Sole primary source
paper.	
'sample_metadata.csv', 'species_profile.csv', '	
strain_candidates.csv', 'strain_abundance_weights.csv'	
', 'genome_catalog.csv'	Reconstruct the sample-
specific truth manifest and expected coverage.	
'pipeline_catalog.csv', 'contigs.csv', '	
contig_genome_mappings.csv', 'bin_assignments.csv'	
Reconstruct representative bins and base-pair	
recovery.	
'contig_coverage_profiles.csv'	Six-sample coverage
vector for every assembler contig.	
'contig_tn_counts.csv'	Raw counts for the 256
tetranucleotides of every contig.	
'genome_tn_profiles.csv'	Normalized 256-component
tetranucleotide profile for each candidate genome.	
'data_validation.json'	Machine-readable integrity and
relationship audit for the exact mounted CSV bytes;
provenance/QC only, not a substitute for analysis. |
```

Identifiers and row order are arbitrary and carry no
ranking information. A contig may map to more
than one truth genome. 'unbinned' is not a bin.

Required analysis

1. Recover and implement the manifest conventions from '
The MAG-E simulator': the
potential-strain screen, the choice within each strain
cluster, the no-potential-strain
fallback, and the allocation of the supplied realized
weights. Compute expected coverage as
sequenced bases x strain relative abundance / genome
length. Keep every manifest row, but
restrict pipeline evaluation to genomes with expected
coverage at least '2.5x'.
2. Recover from 'The MAG-E evaluator' how a representative
bin is selected and how its
whole-genome recall and truth precision are defined.
Apply the same source-defined evaluation
circuit to the prophage and shared-contig strata.
Recover the treatment of multi-mapped contigs
from the paper's 'Contig-level analyses' section.
3. For every contig-to-genome mapping row, recover and
implement the paper's coverage-distance and
tetranucleotide-composition-distance definitions. Apply
each definition within the mapping
row's assembler, so the coverage reference population
for this fixture is an
'(assembler, genome_accession)' population across
samples. The supplied coverage vectors, raw
tetranucleotide counts, and genome profiles are
observations for those calculations; do not
replace them with scalar coverage, GC content, or a
sample-wide profile.
4. Recover the paper's base-pair midpoint percentile
convention and its scientific weighting.
Apply it separately to the mapping-row population from
each assembler. The acceptance-bearing
exact-boundary convention is strict-lower/inclusive-
upper, so an exact boundary stays in the
preceding percentile. Break exact distance ties by '

Task package

| Package files | Scientific and evaluation role |
|--|---|
| <code>instruction.md</code> | Verbatim research question, required analysis, execution command, artifact contract, and answer schema. |
| <code>environment/</code> | Frozen offline Docker environment with pinned dependencies. |
| <code>data/</code> | One paper and fourteen visible data and quality-control files. |
| <code>solution/</code> | Expert implementation, six output artifacts, report, and calibrated reference decision. |
| <code>tests/rubric.json</code> | Twenty observable scientific-reasoning criteria with a total weight of 70. |
| <code>tests/check_programmatic.py</code> | Eight weighted checks, including a held-out replay in which every identifier is relabeled. |
| <code>task.toml</code> | Fixed 0.50 programmatic and 0.50 rubric-based score aggregation. |

Table 13 Contents of an example task package and the role of each component.

| | |
|---|--|
| <p>contig_id', then 'genome_accession', for deterministic replay. Define each high-distance outlier stratum as percentile '>= 80'.</p> <p>5. Within each pipeline/sample, aggregate mapped base pairs, not contig counts or macro means, for the four recovery axes. For an axis, recall is mapped base pairs assigned to that genome's representative bin divided by all mapped base pairs in that stratum. Prophage and shared axes use 'region_class'; coverage and composition axes use their respective percentile. Separately macro-average truth precision and whole-genome recall over eligible genomes within each sample.</p> <p>6. Apply this fixed policy without rounding intermediate values:</p> <ul style="list-style-type: none"> - a pipeline is eligible only if sample-level macro truth precision is '>= 0.90' in every evaluable sample; - for each pipeline, average each of the four base-pair recovery axes across evaluable samples; - define 'robust_recovery_minimum' as the smallest of those four means and name its 'limiting_axis' as 'coverage_outlier', 'composition_outlier', 'prophage', or 'shared'; - among eligible pipelines, maximize 'robust_recovery_minimum'; - exact ties prefer higher global macro whole-genome recall, then higher minimum sample truth precision, then lexicographically smaller 'pipeline_id'; - assign 'policy_rank' as a one-based rank over precision-eligible pipelines in that policy order, and leave 'policy_rank' empty for ineligible pipelines. The fixture has no unresolved tie. <p>7. Separately identify the global whole-genome F-score leader, where F-score is the harmonic mean of global macro truth precision and global macro whole-genome recall, and the global truth-</p> | <p>precision leader. Explain each alternative's observable limiting-axis failure under the policy. Also identify every precision-ineligible pipeline and the sample or samples that fail the floor.</p> <p>Use full-precision calculations. Numeric output is checked with absolute or relative tolerance '1e-6'.</p> <p>## Deliverables</p> <p>Write all files below under '/home/agent/results/'.</p> <pre> Path Required contents --- --- 'code/mage_pipeline_analysis.py' Readable executable implementation of the complete analysis. 'manifest.csv' One row per reconstructed manifest genome. Required: 'sample_id', 'species_cluster', 'strain_cluster', 'genome_accession', 'genome_length_bp', 'strain_relative_abundance', 'expected_coverage_x', 'analysis_eligible'. 'representative_bins.csv' Key 'pipeline_id', 'sample_id', 'genome_accession'. Required: 'expected_coverage_x', 'representative_bin', 'truth_precision', 'whole_genome_recall', 'prophage_assembly_recall', 'shared_assembly_recall'. 'mapping_outlier_metrics.csv' Key 'assembler', 'contig_id', 'genome_accession'. Required: 'sample_id', 'mapped_bp', 'region_class', 'contig_length_bp', 'coverage_distance', 'coverage_distance_percentile', 'tn_distance', 'tn_distance_percentile'. 'sample_pipeline_summary.csv' Key 'pipeline_id', 'sample_id'. Required: 'n_units', 'mean_truth_precision', 'mean_whole_genome_recall', 'coverage_outlier_recall', 'composition_outlier_recall', 'prophage_recall', 'shared_recall', 'precision_floor_met'. 'pipeline_summary.csv' Key 'pipeline_id'. Required: 'assembler', 'binner', 'binning_mode', 'mean_truth_precision', 'mean_whole_genome_recall', 'mean_genome_fscore', 'minimum_sample_truth_precision'</pre> |
|---|--|

```

    'the four 'mean*_recall' axes, '
    robust_recovery_minimum', 'limiting_axis', '
    precision_eligible', and 'policy_rank'. |
| 'answer.json' | Structured headline decision using the
schema below. |
| 'report.md' | Scientific write-up with Context, Data/QC,
Methods, Results, Validation, Limitations, and
Conclusion. Explain the source-derived choices and
why they matter, the two requested leaders, and each
precision-ineligible pipeline with its failing sample
(s). Identify the coverage and tetranucleotide inputs
as reduced semi-synthetic fixture observations,
distinguish the fixture policy from the paper's
conclusions, and do not present the selected pipeline
as a universal paper recommendation. |

```

Additional CSV columns are allowed. Booleans may be case-insensitive 'true'/'false'; ineligible 'policy_rank' values must be empty.

'answer.json' must contain:

```

{
  "dataset_summary": {
    "n_samples": 0, "n_manifest_genomes": 0, "
    n_eligible_sample_genome_units": 0,
    "n_pipelines": 0, "coverage_threshold_x": 2.5, "
    truth_precision_floor": 0.9,
    "outlier_percentile_start": 80
  },
  "manifest_summary": {
    "n_species_without_potential_strain": 0, "
    n_genomes_below_coverage_threshold": 0
  },
  "selected_pipeline": {
    "pipeline_id": "string", "assembler": "string", "
    binner": "string", "binning_mode": "string",
    "mean_truth_precision": 0.0, "mean_whole_genome_recall":
    0.0,
    "minimum_sample_truth_precision": 0.0, "
    mean_coverage_outlier_recall": 0.0,
    "mean_composition_outlier_recall": 0.0, "
    mean_prophage_recall": 0.0, "mean_shared_recall":
    0.0,
    "robust_recovery_minimum": 0.0,
    "limiting_axis": "coverage_outlier|composition_outlier
    |prophage|shared",
    "margin_over_runner_up": 0.0
  },
  "alternative_summary": {
    "whole_genome_fscore_leader_id": "string", "
    whole_genome_fscore_leader_score": 0.0,
    "truth_precision_leader_id": "string", "
    truth_precision_leader_score": 0.0,
    "runner_up_pipeline_id": "string", "
    runner_up_limiting_axis": "string",
    "runner_up_robust_recovery_minimum": 0.0
  },
  "conclusion": {
    "decision": "<PIPELINE_ID><UPPERCASE_LIMITING_AXIS>",
    "selected_pipeline_id": "string",
    "limiting_axis": "string", "confidence": "high|medium|
    low",
    "primary_rationale": "string", "main_limitation": "
    string"
  }
}

```

In 'conclusion.decision', append the uppercase serialization of the selected lowercase 'limiting_axis' token to 'selected_pipeline_id' with one underscore. Keep both standalone 'limiting_axis' fields in the lowercase enum shown above.

Before submitting, confirm that the CLI regenerates all artifacts, percentiles use the recovered base-pair midpoint convention, the source-derived shared-contig handling is applied, every-sample precision is applied before ranking, and all artifacts agree on the same pipeline and limiting axis.

I.2 Environment

The task runs offline in a frozen Docker image. The visible workspace at /home/agent/data/ contains one source paper and fourteen data files (Table 14); no reference artifact or checker input is mounted on the agent side.

I.3 Reference solution

The package ships an expert reference solution (solution/): the analysis code (code/mage_pipeline_analysis.py), all six required CSV/JSON artifacts, and the write-up (report.md, with Context / Data & QC / Methods / Results / Validation / Limitations / Conclusion). Its headline answer.json — the reference decision the checker is calibrated against — is reproduced below. The selected pipeline is MAGE-83, limited by the *composition-outlier* axis; note that this is *neither* the whole-genome F-score leader (MAGE-47) *nor* the truth-precision leader (MAGE-31) — the two decoys the instruction asks the agent to reject.

```

{
  "selected_pipeline": {
    "pipeline_id": "MAGE-83", "assembler": "metaSPAdes", "
    binner": "COMEBin",
    "binning_mode": "multi", "limiting_axis": "
    composition_outlier",
    "mean_truth_precision": 0.9875, "
    mean_whole_genome_recall": 0.3300,
    "minimum_sample_truth_precision": 0.9607,
    "mean_coverage_outlier_recall": 0.5147, "
    mean_composition_outlier_recall": 0.2862,
    "mean_prophage_recall": 0.6123, "mean_shared_recall":
    0.4721,
    "robust_recovery_minimum": 0.2862, "
    margin_over_runner_up": 0.2375
  },
  "alternative_summary": {
    "whole_genome_fscore_leader_id": "MAGE-47", "
    whole_genome_fscore_leader_score": 0.5828,
    "truth_precision_leader_id": "MAGE-31", "
    truth_precision_leader_score": 0.9988,
    "runner_up_pipeline_id": "MAGE-47", "
    runner_up_limiting_axis": "shared",
    "runner_up_robust_recovery_minimum": 0.0488
  },
  "conclusion": {
    "decision": "MAGE-83_COMPOSITION_OUTLIER", "
    selected_pipeline_id": "MAGE-83",
    "limiting_axis": "composition_outlier", "confidence":
    "high",
    "primary_rationale": "Clears the truth-precision floor
    in every evaluable sample and
    maximizes the minimum mean recovery across the four
    axes.",
    "main_limitation": "A deterministic reduced synthetic-
    community fixture, not a full
    replication of the paper's 4,800 tasks."
  },
  "dataset_summary": { "n_samples": 6, "n_manifest_genomes":
    18,
    "n_eligible_sample_genome_units": 17, "n_pipelines": 8,
    "coverage_threshold_x": 2.5,
    "truth_precision_floor": 0.9, "
    outlier_percentile_start": 80 },
  "manifest_summary": { "
    n_species_without_potential_strain": 2,
    "n_genomes_below_coverage_threshold": 1 }
}

```

| Role | Files |
|------------------|---|
| Source & docs | pipelines for metagenome.pdf, data_dictionary.md, data_validation.json |
| Truth / manifest | sample_metadata.csv, species_profile.csv, strain_candidates.csv, strain_abundance_weights.csv, genome_catalog.csv |
| Bins & recovery | pipeline_catalog.csv, contigs.csv, contig_genome_mappings.csv, bin_assignments.csv |
| Contig matrices | contig_coverage_profiles.csv, contig_tn_counts.csv, genome_tn_profiles.csv |

Table 14 The files mounted at /home/agent/data/.

```
}

```

I.4 Grading, part 1 — the LLM rubric (tests/rubric.json)

The judge grades *scientific reasoning and source grounding*, not numeric values: its context (`judge_config.json`) states that “exact identifiers and numeric values are owned by `check_programmatic.py`”. Each criterion is an observable pass/fail statement with named `grader_inputs` (the report, the code, and the specific CSV it must be consistent with); the judge sees the source excerpts and the rubric but never the reference score. The complete rubric is 20 criteria over four categories, total weight 70 (Table 15).

One criterion is shown verbatim to convey the JSON shape, including the calibration bar in its `description` and the artifacts the judge is allowed to read:

```
{ "id": 1, "category": "Source Grounding", "weight": 4,
  "criterion": "Explains why the paper's potential-strain
screen is necessary before
constructing the truth manifest.",
  "description": "Award only when the report explains that
LOW effective lambda makes
adjusted ANI unreliable; potential strains therefore
require non-LOW effective lambda
and adjusted ANI at least 99.8%, plus a 99.5% lower
confidence bound when coverage is
not HIGH. The explanation must connect those
conditions to excluding unreliable strain
matches before truth construction.",
  "source": "pipelines for metagenome.pdf, Methods, The
MAG-E simulator",
  "grader_inputs": ["report.md", "code/
mage_pipeline_analysis.py"] }
```

I.5 Grading, part 2 — the programmatic checker (tests/check_programmatic.py)

The checker recomputes every deliverable from a golden bundle and, crucially, re-runs the agent’s own `mage_pipeline_analysis.py` on a *held-out, re-labeled* copy of the data: all pipeline, genome, and bin identifiers are rewritten (CAL-P01, CAL-G001, CAL-B0001, ...) and the analysis must reproduce the correct decision under the new labels. This `heldout_pipeline_replay` check carries the largest single weight (10 of 30), so hard-coding `MAGE-83` or memorizing any identifier or metric table fails. We list the eight named checks with their weights and the checker’s function names — not their bodies.

```
# --- weighted check registry (CHECK_WEIGHTS; total weight
30) -----
CHECK_WEIGHTS = {
    "primary_selection": 4.0, # selected
    pipeline + limiting axis match
    "manifest_science": 2.0, # manifest
    reconstruction values
    "representative_bin_science": 3.0, # per-genome
    representative-bin metrics
    "mapping_outlier_science": 4.0, # coverage /
    tetranucleotide distances + percentiles
    "sample_recovery_science": 3.0, # four base-pair
    recovery axes, per sample
    "pipeline_policy_science": 3.0, # eligibility,
    maximin ranking, policy_rank
    "dataset_counts": 1.0, # answer.json
    dataset/manifest summaries
    "heldout_pipeline_replay": 10.0, # RERUN under
    relabeled IDs -> anti-hard-coding
}

# --- checker functions (names only)
-----
def _load_csv(...): ... # I/O + golden-
bundle loaders
def _load_bundle(...): ...
def _expected_mapping(...): ...
def _safe_check(...): ... # tolerant
numeric compare (RTOL=ATOL=1e-6)
def _close(...): ...
def _as_bool(...): ...
def _compare_table(...): ...

def _primary_selection(...): ... # the eight
weighted science checks
def _manifest_science(...): ...
```

```

def _representative_bin_science(...): ...
def _mapping_outlier_science(...): ...
def _sample_recovery_science(...): ...
def _pipeline_policy_science(...): ...
def _dataset_counts(...): ...
def _all_science(...): ...

def _heldout_case(data_dir, target): ... # build the
    relabeled (CAL-*) held-out copy
def _rewrite_csv(...): ...
def _heldout_pipeline_replay(results): ... # import +
    rerun agent code on held-out data

def check(results_dir): ...           # aggregate ->
    programmatic score in [0, 1]
def main(): ...

```

The final task reward is $0.50 \cdot (\text{programmatic score}) + 0.50 \cdot (\text{rubric score})$, as declared in `task.toml` for this end-to-end-analysis task.

| # | Category | Weight | Criterion |
|----|----------|--------|---|
| 1 | SG | 4 | Explains why the paper’s potential-strain screen is necessary before constructing the truth manifest. |
| 2 | SG | 3 | Distinguishes the paper’s within-strain-cluster choice from its no-potential-strain fallback. |
| 3 | SG | 3 | Explains the source-defined per-genome representative-bin estimand. |
| 4 | SG | 4 | Explains the paper’s treatment of multi-mapped shared contigs as mapping-level evidence. |
| 5 | SG | 4 | Explains why the paper uses base-pair midpoint percentiles for contig-level analysis. |
| 6 | SC | 4 | Interprets how the expected-coverage eligibility threshold changes the evaluation population. |
| 7 | SC | 4 | Interprets coverage distance as task-local assembler-and-genome-conditioned across-sample deviation. |
| 8 | SC | 4 | Interprets tetranucleotide cosine distance as a composition-shift measure. |
| 9 | SC | 4 | Explains why the four recovery axes are distinct scientific stress tests. |
| 10 | SC | 4 | Explains the consequence of mapped-base aggregation for recovery. |
| 11 | SC | 4 | Justifies the every-sample precision prerequisite as a non-compensatory safeguard. |
| 12 | SC | 4 | Interprets the maximin policy as protection against a weakest-axis failure. |
| 13 | SC | 4 | Explains the selected pipeline’s limiting axis as the decision bottleneck. |
| 14 | SC | 4 | Explains why the whole-genome F-score alternative loses under the declared question. |
| 15 | SC | 4 | Explains why the truth-precision leader is not the robust-recovery choice. |
| 16 | EV | 3 | Uses sample-level evidence to explain every precision-ineligible pipeline. |
| 17 | EV | 3 | Validates the distinction between every-sample eligibility and global precision leadership. |
| 18 | EV | 2 | Justifies the task-local assembler partition without presenting it as a source convention. |
| 19 | CM | 2 | Scopes the pipeline conclusion to the reduced semi-synthetic task fixture. |
| 20 | CM | 2 | Must <i>not</i> claim perturbation, ablation, or sensitivity results absent from the submitted artifacts. |

Table 15 The complete 20-criterion rubric for the full case (total weight 70). Category codes: SG = Source Grounding, SC = Scientific Correctness, EV = Execution & Validation, CM = Communication.