OMICSBENCH: A LONG-HORIZON MULTI-TURN BENCHMARK OF AI AGENTS ON RESEARCH-STUDY-SCALE BIOINFORMATICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Computational biologists increasingly delegate substantial analyses to AI agents in long-horizon, multi-turn sessions, yet most benchmarks of agents' bioinformatics capabilities pose a single short task with a single-shot prompt. We present OmicsBench, a long-horizon, multi-turn bioinformatics benchmark for AI agents. Each of its 97 tasks links dependent subtasks into a directed acyclic graph, for 662 underlying subtasks in all. Each subtask is posed as one user turn and graded deterministically against an answer key, with optional in-context feedback on incorrect answers. Across 12 models on three harnesses, the slowest agent averages 3.6 hours per task, and no combination comes near saturating the benchmark, with mean scores of 0.43 to 0.57 out of a possible 1 without in-context feedback and 0.54 to 0.67 with it. Multi-turn evaluation also uncovers behavior that single-shot prompts cannot, as agents score lower on subtasks that depend on earlier subtasks, and in-context feedback helps agents recover some of that loss. Because deterministic scoring gives no qualitative account of how an agent failed, we also introduce an unsupervised discovery pipeline that describes, embeds, and clusters agents' failed attempts to surface emergent failure modes. Applied to OmicsBench, this pipeline exposes qualitative shortcomings in how agents carry out bioinformatics analyses, ranging from agents applying their own statistical filters in place of the specified ones to agents writing whole answers from memory.

1 INTRODUCTION

AI agents are increasingly capable of carrying out computational biology tasks that require tool use, code execution, and reasoning over biological datasets (Alber et al., 2026; Anthropic, 2026; Nair et al., 2026). Increasingly, computational biologists put these agents to work on long-horizon problems, in which raw data pass through many interdependent steps on the way to a scientific result, each building on the outputs of earlier ones. Such dependencies form a directed acyclic graph (DAG) of analysis steps, the abstraction bioinformaticians already use to define pipelines in workflow systems, such as Snakemake and Nextflow (Di Tommaso et al., 2017; Köster & Rahmann, 2012). Typically, scientists remain in the loop, delegating a substantial analysis, reviewing what comes back and requesting revisions, and only then delegating the next. Most bioinformatics benchmarks, by contrast, pose the whole analysis as a single prompt describing the desired output and grade only what the agent returns. Such benchmarks therefore rarely measure how an agent performs over a realistic workflow, one dependent subtask at a time.

To address this gap, we present OmicsBench, a long-horizon, multi-turn benchmark of AI agents' computational biology capabilities (Figure 1). Each of the 97 tasks in OmicsBench is a DAG of subtasks, 662 in all, that together make up a research-study-scale workflow, with each subtask graded against an answer key and each edge letting a subtask build on the results of earlier ones. Every task is built on a real biological dataset, and its subtasks carry that dataset from raw data through to a scientific result. The tasks are long-horizon, requiring agents to work for hours, with per-agent averages reaching 3.6 hours, 350 tool calls, and 110 million tokens per task. OmicsBench poses each subtask as one user turn of a single session, either with no feedback on its answers ("single-pass mode") or with in-context feedback on a failing answer and a chance to retry ("gated mode").

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

Our contributions are:

- **A long-horizon, multi-turn evaluation framework for bioinformatics.** OmicsBench poses a research-study-scale analysis as a DAG of dependent subtasks. To our knowledge, it is the first bioinformatics benchmark to pose subtasks as the turns of one session and to give in-context feedback. The framework is domain-agnostic, so any research study that is expressible as a DAG of deterministically gradable steps can be implemented as an OmicsBench task, and the benchmark scales to DAGs of arbitrary length and complexity.
- **A benchmark of 97 research-study-scale tasks and 662 subtasks, and an evaluation of 12 frontier models on three harnesses.** Built from published studies and their datasets, OmicsBench uses 58 biologically informed graders that match named entities, such as genes and organisms, via database records rather than matching their names as strings. We release the runner, all graders, and three tasks with their datasets, while the remaining 94 tasks form a private test set. No model/harness combination comes close to saturating the benchmark. We find that agents' scores decline on later turns and dependent subtasks, while in-context feedback helps them recover part of that loss.
- **An unsupervised pipeline for discovering agents' failure modes.** The pipeline describes, embeds, and clusters failed attempts. It surfaces 12 emergent, bioinformatics-specific failure modes beyond those of a priori annotation, ranging from agents applying their own selection criteria to agents fabricating answers from memory. These modes identify concrete behaviors for future models to improve upon and provide valuable post-training signals.

2 Related work

OmicsBench sits between two lines of prior work, short-horizon benchmarks that ask for one small piece of analysis and long-horizon benchmarks that ask for work on the scale of a whole research study, and OmicsBench combines the scale of the latter with the granularity of the former.

2.1 Short-horizon bioinformatics benchmarks

These benchmarks evaluate self-contained pieces of work whose results are not carried into later requests. Some, including LAB-Bench (Laurent et al., 2024), LABBench2 (Laurent et al., 2026), Genome-Bench (Yin et al., 2025), BioProBench (Liu et al., 2026), BioCoder (Tang et al., 2024), BioDSBench (Wang et al., 2025), and CellVoyager's CellBench (Alber et al., 2026), test capabilities such as biological knowledge, protocol reasoning, analysis planning, and code generation, without an agent working with a supplied dataset in a coding environment. Others ask an agent to analyze data in a coding environment and answer a question in one request. These include GenoTEX (Liu et al., 2025), BAISBench (Luo et al., 2026), BiomniBench-DA (Qu et al., 2026), CompBioBench (Nair et al., 2026), BioMysteryBench (Anthropic, 2026), DrugDiscoveryBench (Akyürek et al., 2026), and LatchBio's SpatialBench (Workman et al., 2026b), scBench (Workman et al., 2026a), EpiBench (Muralidharan et al., 2026), VariantBench (Bhowmick et al., 2026), TxBench-PP (Le et al., 2026), TxBench-Antibody Discovery (Ramasamy et al., 2026), and TxBench-Oligonucleotide Discovery (Jacko et al., 2026). Because each item ends after one request, these benchmarks do not measure whether an agent can carry intermediate results through a research-study-scale sequence of dependent subtasks.

2.2 Long-horizon bioinformatics benchmarks

These benchmarks evaluate research-study-scale analyses, but they present the entire objective in a single request. BixBench (Mitchener et al., 2025), BixBench3 (Koch et al., 2026), GeneBench (Li & Ho, 2026), GeneBench-Pro (Li et al., 2026), BioAgent Bench (Fa et al., 2026), BioML-bench (Miller et al., 2025), SpatialBench-Long (Diks et al., 2026a), scBench-Long (Diks et al., 2026b), and the life-science tasks in Terminal-Bench-Science (Terminal-Bench-Science Team, 2026) give an agent data and a research objective, allow it to work without further instruction, and grade its final output against a published result, a reproduced or deterministic ground truth, or an expert rubric. Some of these benchmarks already structure their tasks around dependent steps, such as BixBench3's artifact dependency graph and the cascaded decision points in GeneBench and GeneBench-Pro.

Those dependent steps nevertheless remain inside a single prompt, so the benchmarks neither present successive subtasks as turns in one session nor provide in-context feedback on one step before the next depends on it. Separately, benchmarks that characterize agent failures generally assign attempts to predefined categories, leaving emergent failure modes unobserved.

3 OMICSBENCH

3.1 OMICSBENCH FORMULATION

Each OmicsBench task $T = (P, D, G)$ is composed of a task prompt P, a biological dataset D, and a task DAG $G = (V, E)$, in which each node $v \in V$ is a subtask and each edge $e = (u, v) \in E$ means subtask v depends on the answer to subtask u (Figure 1A). Each subtask $v = (v_p, v_e)$ consists of a subtask prompt v_p and an evaluation set v_e (Table B1). The evaluation set holds one or more deterministic graders g_i, each paired with an answer key y_i, so that $v_e = \{(g_1, y_1), (g_2, y_2), \ldots\}$. An agent answers the subtask prompt by writing an answer file a_v. Each grader in v_e is assigned one field of the answer file a_v and scores that field against its answer key, giving one score per grader, $g_i(a_v, y_i) \in [0, 1]$. A subtask's score s_v is the mean of these grader scores, and the task score S is the mean of all subtask scores,

$$s_v = \frac{1}{|v_e|} \sum_i g_i(a_v, y_i), \qquad S = \frac{1}{|V|} \sum_{v \in V} s_v.$$

To evaluate an agent on a task, we open the session with the benchmark system prompt and the task prompt, which together describe the problem and the dataset (Appendix C.1), and the task DAG is then unrolled in dependency order, with every subtask prompt becoming a user turn in that same continuous session (Appendix C.2). Because the session is continuous, the agent's previous answer files stay on hand in the execution container, and its reasoning for those answers stays in its context. A turn ends when the agent stops working or when a time limit $t_{\max}$ is reached, whichever comes first, and whatever answer file is on disk at that moment is the one graded.

In single-pass mode, the agent receives no feedback on its answers, and when it stops working, the next subtask prompt follows (Figure 1B). In gated mode, an answer scoring below a gate threshold τ is returned to the agent as a failure, and it is re-prompted to fix its work (Appendix C.3) until an answer scores at or above τ, or it reaches an attempt limit θ (Figure 1C, Table B2). In either mode, a subtask's recorded score is that of its best attempt, and when the best attempt was not the last, the workspace is restored to it before the next subtask is posed (Appendix C.2). Single-pass mode measures what an agent can produce unaided, and gated mode measures how much in-context feedback allows it to recover its score (Figure 2B).

3.2 BENCHMARK CONSTRUCTION

OmicsBench tasks are built from published papers and their accompanying biological datasets by a pipeline of agents and of human experts with graduate-level computational biology training (Figure 2A). Human experts first curate papers, whose reported results serve as ground truth, and their accompanying biological datasets. A writer agent then uses the papers and datasets to draft OmicsBench tasks, as in BixBench3 (Koch et al., 2026). A reviewer agent confirms that the tasks are supported by their papers and free of data leakage, or sends them back to the writer, rejecting any task that is unfixable or still flawed after three rounds of revision. Automated quality checks confirm that each task is in valid OmicsBench format, and human experts then review and edit each task, or reject it. Three of the 97 tasks are publicly released, and the remaining 94 are held back as a private test set to keep the benchmark useful for evaluating future models (Appendix E).

The task datasets span a wide range of data modalities, with transcriptomic assays in 66 tasks, proteomic assays in 29, and epigenomic assays in 9, with 21 tasks drawing on more than one modality (Figure 3A). The datasets come from ten different organisms (Figure 3D), have a median of 3 files and a maximum of 32 (Figure 3G), and range in size from 365 KB to 8.4 GB (Figure 3H). The biology that the tasks address covers 11 biological domains (Figure 3B), and the subtasks cover 18 common bioinformatics analysis types (Figure 3C). Most tasks have between 4 and 10 subtasks, with a median of 7 (Figure 3E), and in the median task the longest chain of dependent subtasks runs to 3 (Figure 3F), with many tasks featuring branching and joining structures in their DAGs.

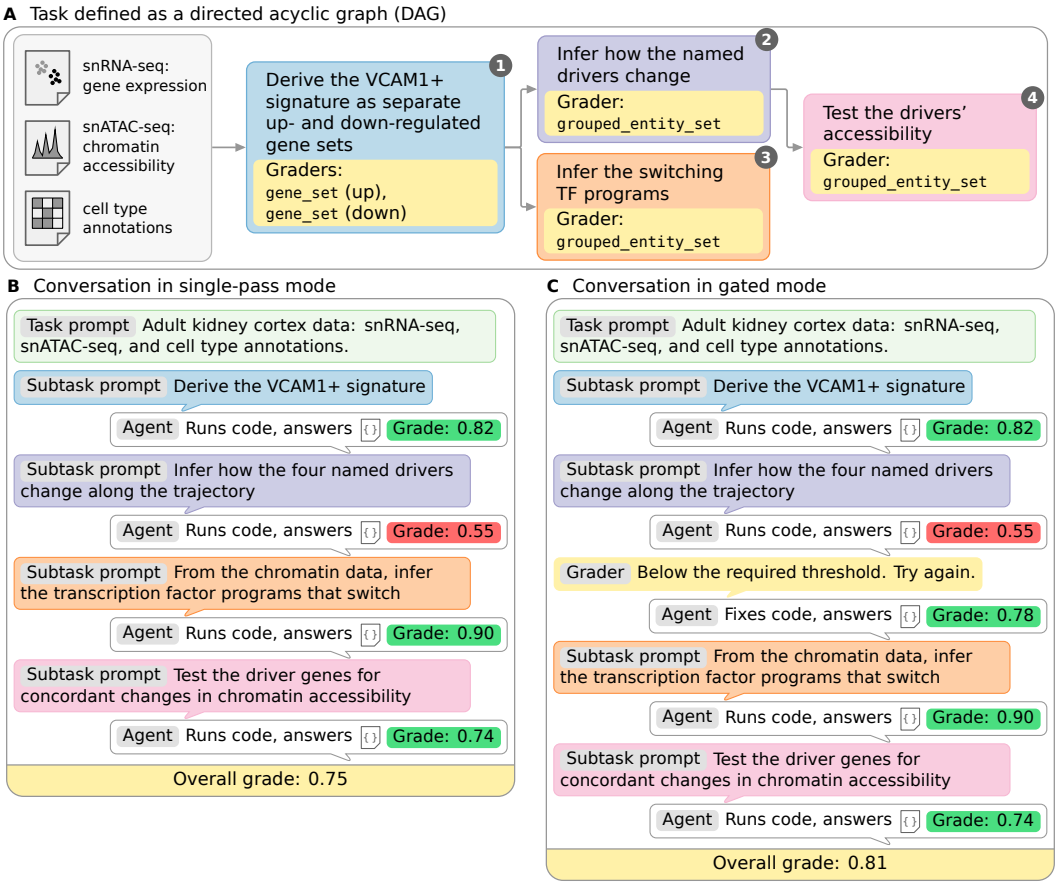

Figure 1: **Example OmicsBench task.** A stylized view of the example task in Appendix D (Table D1, Table D2). (**A**) A task is a DAG of analysis subtasks on a dataset. (**B**) Single-pass mode, with each subtask one user turn and no feedback on answers. (**C**) Gated mode, where an answer scoring below the gate threshold (τ) is returned for a retry in the same session, until an answer reaches τ or an attempt limit.

3.3 Grading

Because biological results take varied forms and rarely reduce to single values compared for equality, OmicsBench uses biologically informed graders, each built for one kind of answer (Table F1). For example, two correct enrichment analyses may order their terms differently and report slightly different p-values, so the `enrichment_result_set` grader requires the terms to match, compares p-values on a log scale, and ignores order. Each of the other 57 graders likewise checks what is biologically meaningful in its kind of answer and ignores the rest, with equivalent behavior for biological data types ranging from dose-response curves to genomic intervals to docking poses.

Furthermore, many types of biological entities are referred to by multiple identifiers. For example, the gene CD274 is also known as PD-L1 and B7-H1, and a string match would mark any of the three valid identifiers wrong against an answer key that uses another. OmicsBench graders therefore match genes, organisms, metabolites, microRNAs, and antimicrobial resistance genes through a reference database for each kind of entity rather than by exact string matching (Table G1).

3.4 Failure mode analysis

We characterized failed attempts with both a priori failure mode annotation and an unsupervised failure mode discovery pipeline that surfaces emergent failure modes in agents' biological reasoning (Figure 2B). For both, we drew a single sample of 150 of the 662 subtasks, stratified by turn position

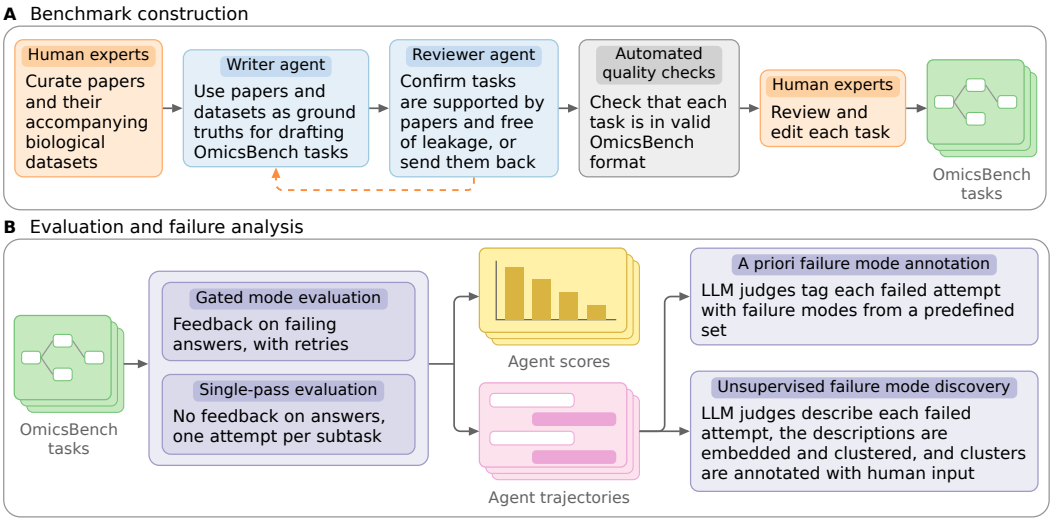

Figure 2: **Construction and evaluation of OmicsBench.** (**A**) Benchmark construction. (**B**) Evaluation and failure analysis pipeline.

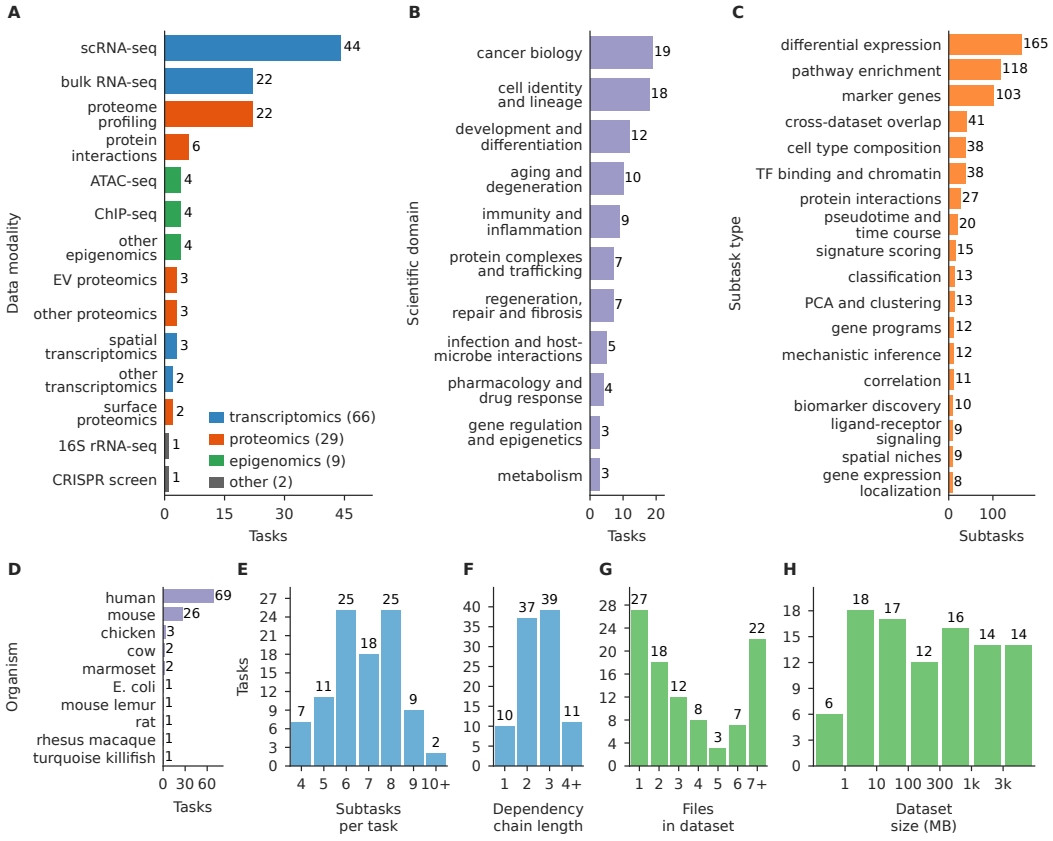

Figure 3: **OmicsBench task statistics.** (**A**) Data modality, with a task counted once per modality it includes. (**B**) Scientific domain, one per task. (**C**) Subtask type, one per subtask. (**D**) Organism, with a task counted once per organism. (**E**) Number of subtasks per task. (**F**) Length of the longest dependency chain in each task's DAG. (**G**) Number of files in each task's dataset. (**H**) Total size of each task's dataset in MB, in log-spaced bins.

and by the kind of result the subtask asks for, and analyzed every attempt scoring below τ at those subtasks. We used only single-pass trajectories, so that every failed attempt is an agent's first and only try at a subtask. All LLM judges were Opus 5 at temperature 0 (Appendix L).

Our a priori vocabulary of 13 failure modes is composed of the ten defined in the BixBench3 study (Koch et al., 2026) as well as three we added for answer sets that are too broad, too strict, or selected on the wrong criterion (Table H1). For each failed attempt, an LLM judge wrote a short narrative of the approach the agent took and how that approach produced the submitted answer, and assigned as many failure modes from the vocabulary as the evidence supported (Appendix L.1). Repeated annotation passes of the LLM judge agreed with a Cohen's kappa of 0.76 (Appendix H).

The unsupervised failure mode discovery pipeline that we built describes, embeds, and clusters failed attempts to group those that failed in the same way, surfacing emergent failure modes that a predefined set would miss. An LLM judge writes a short narrative of each failed attempt and rewrites it as a task-agnostic description of how the submitted answer diverged from the answer key (Appendix L.2). The pipeline then embeds the descriptions, reduces them with UMAP (McInnes et al., 2018), and clusters them with HDBSCAN (Campello et al., 2013), which groups only the points that lie in dense regions and leaves the rest unassigned, so it does not force every attempt into a cluster. A separate LLM judge writes a name and definition for each cluster from the descriptions nearest its center (Appendix L.3). We edited every name and definition by hand for readability and grouped the failure modes into five families by the mechanism of failure they share (Appendix I).

4 Experiments

4.1 Setup

For this evaluation, we set the gate threshold τ to 0.7, the attempt limit θ to 3, the time limit per turn $t_{\max}$ to 1800 s, and the number of runs per task in each mode k to 3 (Table B2). We evaluated 12 models on three harnesses, with each harness pinned to one version and with reasoning effort set to high wherever the model allowed it (Appendix J, Table J1). All 12 models ran on OpenCode, and Opus 5 and GPT-5.6 Sol also ran on their native harnesses, Claude Code and Codex, for 14 model/harness combinations. Of the three pairs of harnesses, two share a single model each and the third shares none, so we cannot make any claims as to the effect of different harnesses on performance. Every run took place in its own Docker container with no network access (Appendix K).

We treat the 97 tasks as a sample from a larger population of bioinformatics tasks, so the uncertainty we report is how much a mean could move had a different 97 been drawn. Because runs of the same task are not independent draws from that population, we averaged each task's three runs into one score before computing any statistic, as counting them separately would understate the uncertainty (Miller, 2024). Every 95% confidence interval is the mean plus or minus 1.96 standard errors, computed from the sample standard deviation of the 97 task scores.

4.2 Evaluation results

No model/harness combination comes close to saturating OmicsBench in either mode, with the best, Opus 5 on OpenCode, averaging 0.57 in single-pass mode and 0.67 in gated mode, and the weakest, Grok 4.3, averaging 0.43 and 0.54 (Figure 4A). On their native harnesses, Opus 5 and GPT-5.6 Sol each score within 0.02 of their OpenCode results. Gated retries raise every combination's score, by 0.09 to 0.13. Within each combination, task scores span nearly the full range (Figure A1A), and the three runs of a task vary little from one another (Figure A2). Task difficulty is spread rather than bunched, running from 0.24 to 0.94 in single-pass mode (Figure 4B), and gated retries lift the whole distribution without flattening it (Figure 4C, Figure A1). Agents drew on 103 distinct domain packages spanning Python, R, and command-line ecosystems, with a median of 2 to 9 per task depending on the combination (Figure A4, Figure A5).

4.3 Cost and efficiency

The model/harness combinations that we tested separate far more by efficiency than by performance (Figure 5). In single-pass mode, Gemini 3.1 Pro costs 12 times what MiniMax-M3 does while

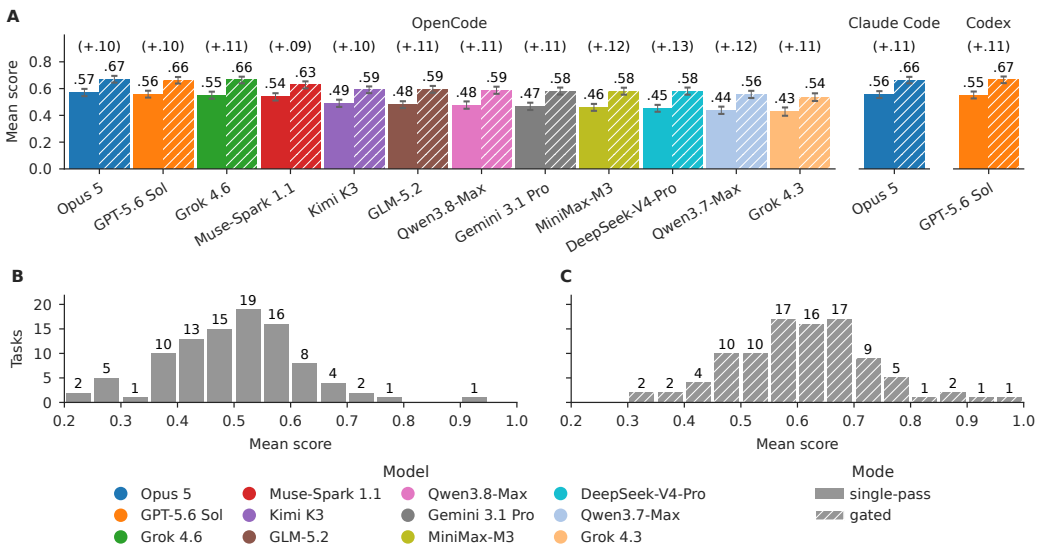

Figure 4: **Performance of the 12 models and three harnesses.** (**A**) Mean score per combination in each mode, with the gated gain in parentheses above each pair. Error bars are 95% confidence intervals over the 97 tasks. (**B, C**) Task difficulty, a task's mean score over all combinations, in each mode.

scoring only 0.01 higher, and 7 times what Muse-Spark 1.1 does while actually scoring 0.07 lower (Figure 5A). Effort has a similar relationship with performance, with Gemini 3.1 Pro making more tool calls and using more tokens[1] than any other combination in both modes while still scoring in the bottom half of the combinations on OpenCode (Figure 5C, Figure 5D, Figure A3). To the extent that OmicsBench represents the work it models, choosing an agent for it is as much an economic question as a capability one. For each measure (cost, time, tool calls, or tokens), a Pareto-optimal model is defined as one that no other model on OpenCode outscores at an equal or lower value of that measure. By this definition, Opus 5 and Grok 4.3 are Pareto-optimal on all four measures in both modes, as is Muse-Spark 1.1 in single-pass mode, while Gemini 3.1 Pro, GLM-5.2, and Qwen3.8-Max are optimal on none of the measures for which they have data (Figure 5, Table A1).

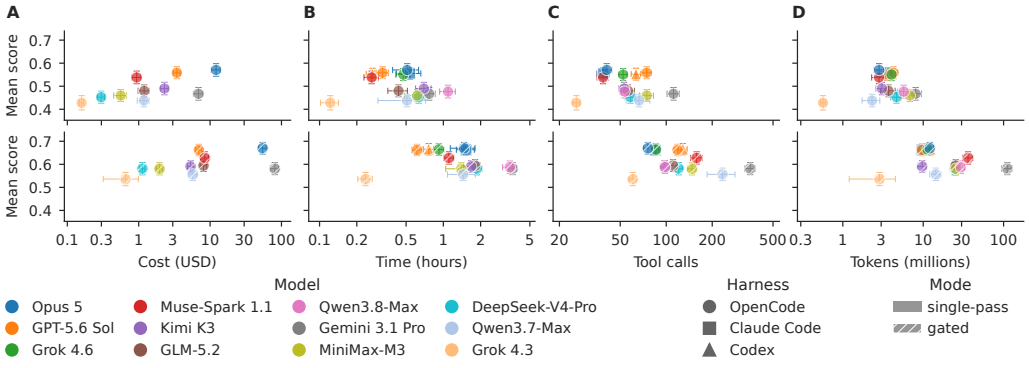

Figure 5: **Cost and efficiency.** (**A**) Cost, (**B**) wall-clock time, (**C**) tool calls, and (**D**) tokens per task against mean score, in single-pass mode (top) and gated mode (bottom). Cost is available only on OpenCode, and not for Grok 4.6 or Qwen3.8-Max. Error bars are 95% confidence intervals of mean score.

[1]Token counts may not be directly comparable between harnesses, since each harness records token use in its own way, internally.

4.4 Performance across turns, dependencies, and retries

An agent's score on a subtask depends on where the subtask falls in its session. Every combination loses 0.06 to 0.12 by the second turn, and later turns stay below the first in all but one case (Figure 6A). Subtasks that build on an agent's earlier answers also score lower than those with no upstream dependence, for every combination (Figure 6B). Gated retries recover part of that loss, as every combination's best score so far rises with each attempt, most from the first attempt to the second (Figure 6C). Even so, only 22 to 34% of subtasks reach the gate within three attempts, up from 15 to 24% on the first attempt (Figure 6D). These analyses leave out the 177 of 662 subtasks graded only by a closed-choice grader (Table F1), because we anecdotally saw that agents tended to exhaust their options by guessing, which we regard as a sign of reward hacking.

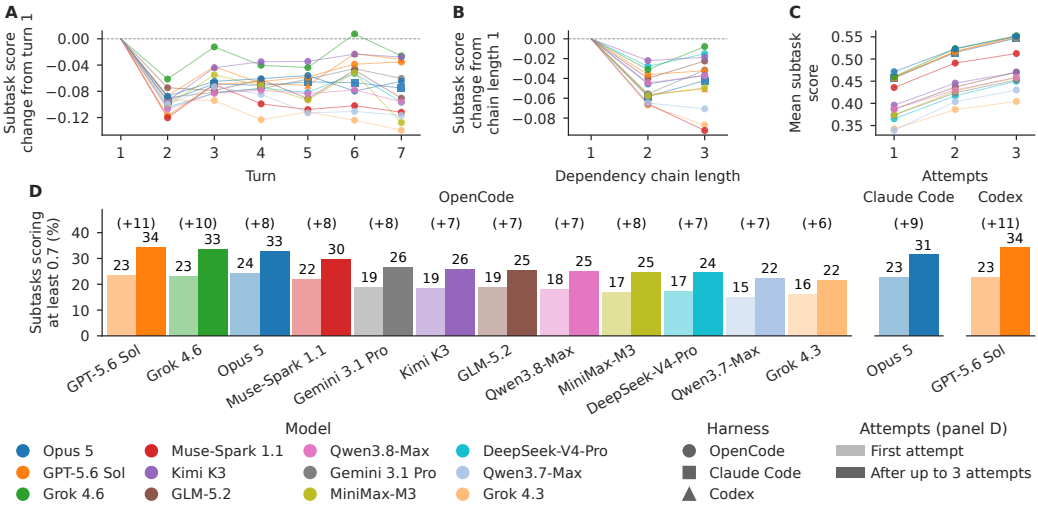

Figure 6: **Subtask scores across a session and across gated retries.** (**A**) Change in mean subtask score from the first turn, in single-pass mode, through turn 7. (**B**) The same by dependency chain length, through length 3, as too few subtasks have longer chains for a stable average. (**C**) Mean best score within 1 to 3 gated attempts. (**D**) Share of subtasks reaching the gate ($\tau = 0.7$) on the first attempt and within three attempts.

4.5 Failure modes

The a priori failure mode distribution shows that, across all model/harness combinations, agents mostly fail in how they carry out the analysis rather than by failing to deliver an answer (Table H1, Figure 7A). Set-selection failures, in which the agent submits far more or far fewer results than the subtask requires, are the most common in every combination, at 25 to 71% of assigned failure modes. Grok 4.3 alone departs from this pattern, with synthetic or placeholder output at 20% and writing no answer at 13% of its assigned failure modes, against at most 7% and 5% for any other combination. Where other agents filter their results poorly, Grok 4.3 more often fabricates them or gives up.

The 12 emergent failure modes cover 42% of failed attempts and fall into five families of related mechanisms (Figure 7C, Section 3.4, Table I1), and most of the remaining 58% are one-off mechanisms too rare to form a cluster, with the rest in the three clusters we set aside (Appendix I). The families have similar shares across combinations (Figure 7B). The largest, at 37 to 55% of a combination's clustered attempts, is submitting more than the subtask asked for (1–4), for example by returning a fixed number of top genes for every group (1), and the next, at 22 to 34%, is substituting the agent's own criterion for the subtask's (6–8), for example by testing only a list of candidate genes the agent wrote itself rather than the whole dataset (7). The other three are substituting a different procedure (11–12, 10 to 16%), putting prior knowledge in place of the data (9–10, 3 to 15%), and submitting less than the subtask requested (5, 4 to 12%). Grok 4.3 again departs from this pattern and is left out of these ranges, as writing the answer from memory (10) alone is half of its clustered attempts, against at most 7% for any other combination.



Figure 7: **Failure modes, by model/harness combination.** (**A**) Each a priori mode's share of a combination's assigned modes. (**B**) Each emergent mode's share of a combination's clustered attempts. (**C**) UMAP of failure descriptions, one dot per attempt, colored by mode, gray if unassigned.

5 Discussion

Implications for training. OmicsBench points to three behaviors worth training toward, namely applying a subtask's stated criterion rather than one of the agent's own (Section 4.5), staying grounded in the scientific context across a session, and using in-context feedback well (Section 4.4).

Limitations. With 97 tasks across 11 biological domains and 18 subtask analysis types, each domain or analysis type holds too few tasks to compare models within it (Figure 3B, Figure 3C). Only Opus 5 and GPT-5.6 Sol ran on more than one harness, so the effect of the harness cannot be separated from that of the model. Lastly, the LLM judge was tested for reproducibility but not against human annotators, and the reward hacking we observed went unquantified.

Future work. The clearest next step for long-horizon, multi-turn evaluation is replacing a fixed sequence of prompts with a simulated user that interacts with agents as a scientist would, though such a user must be held to accurate replies, and an evaluator that writes its own prompts is no longer deterministic. Additionally, more tasks in each domain and analysis type, human annotation of failed attempts, and a quantitative study of reward hacking would address the limitations above.

6 Conclusion

We introduced OmicsBench, 97 research-study-scale bioinformatics tasks whose 662 subtasks are posed as the turns of one session per task, with its runner and graders released under the MIT license and three tasks under CC BY 4.0. No combination of the 12 models and three harnesses comes close to saturating it, as every combination scores lower after its first turn and on dependent subtasks, gated retries recover only part of that loss, and agents follow their own habits over the task's instructions.

AI USE STATEMENT

We used generative AI tools in the following parts of this study.

- **Code.** Agentic coding tools (e.g., Claude Code) helped write the code for this study, including the evaluation runner, the graders, and the analysis and figure scripts, with a human steering the work and reading all of the code.
- **Benchmark construction.** LLM agents drafted and reviewed candidate tasks from published papers and their datasets, under human expert curation and final review (Section 3.2).
- **Evaluation.** The agents evaluated in Section 4 are AI systems.
- **Failure mode analysis.** LLM judges annotated failed attempts, wrote the failure descriptions that were clustered, and assisted in naming the clusters (Section 3.4, Appendix L), with the judges' annotations tested for reproducibility and every failure mode name and definition edited by hand.
- **Writing.** Generative AI tools were used to polish the writing of this paper for readability, and an author checked every claim and citation.

We reviewed all AI-assisted work and take responsibility for the final content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

OmicsBench tasks are built from published studies and the datasets their authors released publicly. We collected no new data from human participants; the human-derived datasets we use are the de-identified public releases of the original studies, and we redistribute data only for the three public tasks, under the original licenses. The human experts who curated papers and reviewed tasks have graduate-level training in computational biology, were compensated for their work at above-market rates, and were informed of how it would be used. The benchmark evaluates AI systems only. We hold 94 of the 97 tasks private to limit contamination of future models' training data. The tasks reproduce analyses already in the literature and confer no capability beyond what the source papers describe, so we see no dual-use concern.

REPRODUCIBILITY STATEMENT

The OmicsBench runner and its 58 graders are open source, and public tasks are released with their datasets, answer keys, and graders, together with the identifier databases against which the graders match, so the evaluation pipeline can be run and audited end to end (Appendix E). The prompts every agent received are reproduced in Appendix C and the LLM judge prompts in Appendix L. The benchmark runtime environment details, resource limits, and time limits are given in Appendix K, and the model and harness versions in Appendix J. The replicate design and every interval and test in the paper follow Section 4.1.

REFERENCES

Afra Feyza Akyürek, Xinming Tu, Alec Gutmanstein, Jason Qin, Divyansh Agarwal, Sergey Chekhov, Brenda Hernandez Villegas, Kirill Chugunov, Judah Engel, Veronica Chatrath, Oscar Kavanagh, Geobio Boo, Ernesto Hernandez, Ying Liu, Emily Xue, Aakash Sabharwal, Daniel Yue Zhang, Zainab Doctor, Yuanhao Qu, and Yunzhong He. DRUGDISCOVERYBENCH: Can Coding Agents Assist Early-Stage Drug Discovery?, 2026.

Samuel Alber, Bowen Chen, Eric Sun, Alina Isakova, Aaron J. Wilk, and James Zou. CellVoyager: AI CompBio agent generates new insights by autonomously analyzing biological data. *Nature Methods*, 23(4):749–759, April 2026. ISSN 1548-7105. doi: 10.1038/s41592-026-03029-6. URL https://www.nature.com/articles/s41592-026-03029-6.

Brian P. Alcock, William Huynh, Romeo Chalil, Keaton W. Smith, Amogelang R. Raphenya, Mateusz A. Wlodarski, Arman Edalatmand, Aaron Petkau, Sohaib A. Syed, Kara K. Tsang, Sheridan J. C. Baker, Mugdha Dave, Madeline C. McCarthy, Karyn M. Mukiri, Jalees A. Nasir,

Bahar Golbon, Hamna Imtiaz, Xingjian Jiang, Komal Kaur, Megan Kwong, Zi Cheng Liang, Keyu C. Niu, Prabakar Shan, Jasmine Y. J. Yang, Kristen L. Gray, Gemma R. Hoad, Baofeng Jia, Timsy Bhando, Lindsey A. Carfrae, Maya A. Farha, Shawn French, Rodion Gordzevich, Kenneth Rachwalski, Megan M. Tu, Emily Bordeleau, Damion Dooley, Emma Griffiths, Haley L. Zubyk, Eric D. Brown, Finlay Maguire, Robert G. Beiko, William W. L. Hsiao, Fiona S. L. Brinkman, Gary Van Domselaar, and Andrew G. McArthur. CARD 2023: expanded curation, support for machine learning, and resistome prediction at the Comprehensive Antibiotic Resistance Database. *Nucleic Acids Research*, 51(D1):D690–D699, January 2023. ISSN 1362-4962. doi: 10.1093/nar/gkac920.

Anthropic. Evaluating Claude's bioinformatics research capabilities with BioMysteryBench, 2026. URL https://www.anthropic.com/research/Evaluating-Claude-For-Bioinformatics-With-BioMysteryBench.

Aashka Bhowmick, Nasim Rahmatpour, Shivang Sharma, Qian Xu, Ananya Gupta, Asmita Lagwankar, Arjun Banerjee, and Christopher Zou. VariantBench: An Agentic Benchmark for Genetic Variant Discovery and Interpretation, 2026. URL https://latch.bio/variantbench.

Garth R. Brown, Vichet Hem, Kenneth S. Katz, Michael Ovetsky, Craig Wallin, Olga Ermolaeva, Igor Tolstoy, Tatiana Tatusova, Kim D. Pruitt, Donna R. Maglott, and Terence D. Murphy. Gene: a gene-centered information resource at NCBI. *Nucleic Acids Research*, 43(D1):D36–D42, January 2015. ISSN 0305-1048. doi: 10.1093/nar/gku1055. URL https://doi.org/10.1093/nar/gku1055.

Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu (eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 160–172, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-37456-2. doi: 10.1007/978-3-642-37456-2_14.

Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, April 2017. doi: 10.1038/nbt.3820. URL https://www.nature.com/articles/nbt.3820.

Ian Diks, Harihara Muralidharan, Tim Proctor, and Kenny Workman. Verifiable Benchmarking of Long-Horizon Spatial Biology, May 2026a. URL http://arxiv.org/abs/2605.28065. arXiv:2605.28065 [cs.AI].

Ian Diks, Zhen Yang, Arjun Banerjee, Tim Proctor, and Kenny Workman. scBench-Long: Verifiable Benchmarking of Long-Horizon Single-Cell Biology, June 2026b. URL http://arxiv.org/abs/2606.26563. arXiv:2606.26563 [q-bio.GN].

Dionizije Fa, Marko Čuljak, Bruno Pandža, and Mateo Čupić. BioAgent Bench: An AI Agent Evaluation Suite for Bioinformatics. In *Proceedings of the 43rd International Conference on Machine Learning*, June 2026. URL https://openreview.net/forum?id=2iFauBXg7Y.

Martin Jacko, Jackson Brougher, Alex Urrutia, Hannah Le, Arjun Banerjee, Dillon Flood, and Kenny Workman. TxBench-Oligonucleotide Discovery: Benchmarking AI Agents on Experimentally Grounded Decisions in the Discovery of ASO/siRNA Therapeutics, 2026. URL https://latch.bio/txbench-oligo.

Zane Koch, Asmamaw T. Wassie, Javier Valdes-Aleman, Jason Lee, Michaela M. Hinks, Samuel G. Rodriques, Andrew D. White, and Jon M. Laurent. BixBench3: Benchmarking AI agents on research-study-scale computational biology tasks, August 2026. URL http://arxiv.org/abs/2608.25286. arXiv:2608.25286 [cs.AI].

Ana Kozomara, Maria Birgaoanu, and Sam Griffiths-Jones. miRBase: from microRNA sequences to function. *Nucleic Acids Research*, 47(D1):D155–D162, January 2019. ISSN 0305-1048. doi: 10.1093/nar/gky1141. URL https://doi.org/10.1093/nar/gky1141.

Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, October 2012. doi: 10.1093/bioinformatics/bts480. URL https://doi.org/10.1093/bioinformatics/bts480.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. LAB-Bench: Measuring Capabilities of Language Models for Biology Research, July 2024. URL http://arxiv.org/abs/2407.10362. arXiv:2407.10362 [cs.AI].

Jon M. Laurent, Albert Bou, Michael Pieler, Conor Igoe, Alex Andonian, Siddharth Narayanan, James Braza, Alexandros Sanchez Vassopoulos, Jacob L. Steenwyk, Blake Lash, Andrew D. White, and Samuel G. Rodriques. LABBench2: An Improved Benchmark for AI Systems Performing Biology Research, May 2026. URL http://arxiv.org/abs/2604.09554. arXiv:2604.09554 [cs.AI].

Hannah Le, Ramesh Ramasamy, Alex Urrutia, Mahsa Yazdani, Tim Proctor, and Kenny Workman. TxBench-PP: Analyzing AI Agent Performance on Small-Molecule Preclinical Pharmacology, June 2026. URL http://arxiv.org/abs/2606.19245. arXiv:2606.19245 [cs.AI].

Jeremy Li and Andrew Ho. GeneBench: Assessing AI Agents for Multi-Stage Inference Problems in Genomics and Quantitative Biology, April 2026. URL https://www.biorxiv.org/content/10.64898/2026.04.22.720113v1.

Jeremy Li, Suyash Shringarpure, Edmund Wong, and Andrew Ho. GeneBench-Pro: Evaluating Multistage Statistical Reasoning in Genomics, Quantitative Biology, and Translational Biomedicine, July 2026. URL https://www.biorxiv.org/content/10.64898/2026.06.29.735386v4.

Haoyang Liu, Shuyu Chen, Ye Zhang, and Haohan Wang. GenoTEX: An LLM Agent Benchmark for Automated Gene Expression Data Analysis. In *Machine Learning in Computational Biology (MLCB 2025)*, 2025. doi: 10.48550/arXiv.2406.15341. URL http://arxiv.org/abs/2406.15341. arXiv:2406.15341 [cs.LG].

Yuyang Liu, Liuzhenghao Lv, Xiancheng Zhang, Jingya Wang, Li Yuan, and Yonghong Tian. BioProBench: A Corpus and Benchmark for Biological Protocol Reasoning in Autonomous Science. In *Proceedings of the 43rd International Conference on Machine Learning*, June 2026. URL https://openreview.net/forum?id=8EZamwNqld.

Erpai Luo, Jinmeng Jia, Yifan Xiong, Xiangyu Li, Xiaobo Guo, Baoqi Yu, Minsheng Hao, Lei Wei, and Xuegong Zhang. Benchmarking AI scientists for omics data-driven biological discovery, January 2026. URL http://arxiv.org/abs/2505.08341. arXiv:2505.08341 [cs.AI].

Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, 2018. URL https://arxiv.org/abs/1802.03426.

Evan Miller. Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations, 2024. URL https://arxiv.org/abs/2411.00640.

Henry E. Miller, Matthew Greenig, Benjamin Tenmann, and Bo Wang. BioML-bench: Evaluation of AI Agents for End-to-End Biomedical ML, September 2025. URL https://www.biorxiv.org/content/10.1101/2025.09.01.673319v3.

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, October 2025. URL http://arxiv.org/abs/2503.00096. arXiv:2503.00096 [q-bio].

Harihara Muralidharan, Reema Baskar, Soo Hee Lee, Tim Proctor, and Kenny Workman. EpiBench: Verifiable Evaluation of AI Agents on Epigenomics Analysis, June 2026. URL http://arxiv.org/abs/2606.13602. arXiv:2606.13602 [cs.AI].

Surag Nair, Laura Gunsalus, Brian Orcutt-Jahns, Jordan Rossen, Avantika Lal, Carlo De Donno, Muhammed Hasan Çelik, Kipper Fletez-Brant, Xiaoman Xie, Hector Corrada Bravo, and Gokcen Eraslan. Agentic systems are adept at solving well-scoped, verifiable problems in computational biology, May 2026. URL https://www.biorxiv.org/content/10.64898/2026.04.06.716850v2.

Yuanhao Qu, Yingzhou Lu, Xinming Tu, Serena Zhang, Tianwei She, Alexander Glenn Shaw, Jou-Ho Shih, Bingqing Zhao, Minjie Shen, Haochen Yang, Jielin Yan, Rongchuan Zhang, Xinze Wu, Tingting Li, Bin Zhou, Ning Wang, Adam Ma, Le Cong, Xiaobo Hu, Yuan Jiang, Jiayun Dong, Tao Peng, Jure Leskovec, and Kexin Huang. BiomniBench: Process-level Evaluation of LLM Agents for Real-world Biomedical Research, May 2026. URL https://www.biorxiv.org/content/10.64898/2026.05.12.724604v2.

Ramesh Ramasamy, Hannah Le, Jackson Brougher, Alex Urrutia, Arjun Banerjee, Sean Poust, and Kenny Workman. TxBench-Antibody Discovery: Benchmarking AI Agents on Experimentally Grounded Decisions in Therapeutic Antibody Discovery, 2026. URL https://latch.bio/txbench-ab.

Conrad L Schoch, Stacy Ciufo, Mikhail Domrachev, Carol L Hotton, Sivakumar Kannan, Rogneda Khovanskaya, Detlef Leipe, Richard Mcveigh, Kathleen O'Neill, Barbara Robbertse, Shobha Sharma, Vladimir Soussov, John P Sullivan, Lu Sun, Seán Turner, and Ilene Karsch-Mizrachi. NCBI Taxonomy: a comprehensive update on curation, resources and tools. *Database*, 2020: baaa062, January 2020. ISSN 1758-0463. doi: 10.1093/database/baaa062. URL https://doi.org/10.1093/database/baaa062.

Xiangru Tang, Bill Qian, Rick Gao, Jiakang Chen, Xinyun Chen, and Mark B Gerstein. BioCoder: a benchmark for bioinformatics code generation with large language models. *Bioinformatics*, 40(Supplement_1):i266–i276, July 2024. doi: 10.1093/bioinformatics/btae230. URL https://doi.org/10.1093/bioinformatics/btae230.

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.

Zifeng Wang, Benjamin Danek, Ziwei Yang, Zheng Chen, and Jimeng Sun. Can Large Language Models Replace Data Scientists in Biomedical Research?, April 2025. URL http://arxiv.org/abs/2410.21591. arXiv:2410.21591 [cs.AI].

David S. Wishart, AnChi Guo, Eponine Oler, Fei Wang, Afia Anjum, Harrison Peters, Raynard Dizon, Zinat Sayeeda, Siyang Tian, Brian L. Lee, Mark Berjanskii, Robert Mah, Mai Yamamoto, Juan Jovel, Claudia Torres-Calzada, Mickel Hiebert-Giesbrecht, Vicki W. Lui, Dorna Varshavi, Dorsa Varshavi, Dana Allen, David Arndt, Nitya Khetarpal, Aadhavya Sivakumaran, Karxena Harford, Selena Sanford, Kristen Yee, Xuan Cao, Zachary Budinski, Jaanus Liigand, Lun Zhang, Jiamin Zheng, Rupasri Mandal, Naama Karu, Maija Dambrova, Helgi B. Schiöth, Russell Greiner, and Vasuk Gautam. HMDB 5.0: the Human Metabolome Database for 2022. *Nucleic Acids Research*, 50(D1):D622–D631, January 2022. ISSN 1362-4962. doi: 10.1093/nar/gkab1062.

Kenny Workman, Zhen Yang, Harihara Muralidharan, Aidan Abdulali, and Hannah Le. scBench: Evaluating AI Agents on Single-Cell RNA-seq Analysis, February 2026a. URL http://arxiv.org/abs/2602.09063.

Kenny Workman, Zhen Yang, Harihara Muralidharan, and Hannah Le. SpatialBench: Can Agents Analyze Real-World Spatial Biology Data?, January 2026b. URL http://arxiv.org/abs/2512.21907. arXiv:2512.21907 [cs.AI].

Ming Yin, Yuanhao Qu, Ling Yang, Le Cong, and Mengdi Wang. Toward Scientific Reasoning in LLMs: Training from Expert Discussions via Reinforcement Learning, June 2025. URL http://arxiv.org/abs/2505.19501. arXiv:2505.19501 [cs.AI].

A SUPPLEMENTARY FIGURES AND TABLES

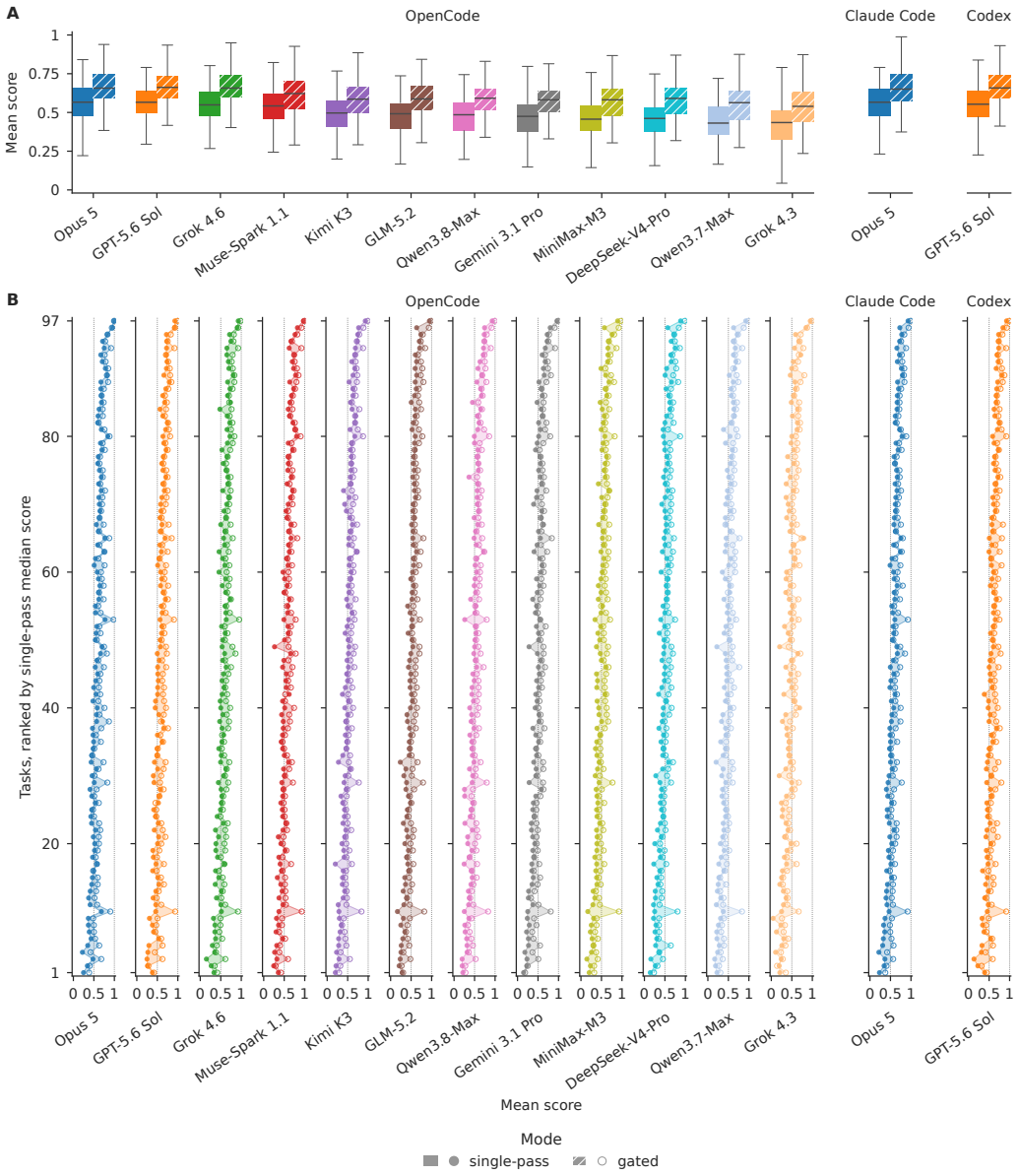

Figure A1: **Per-task scores by model/harness combination.** (**A**) Distribution of the 97 task scores per combination in each mode. (**B**) One row per task, ranked by median single-pass score across combinations, hardest at the bottom. The colored band between dots is tinted where gating raises that task's score for that combination and left white where it lowers it.

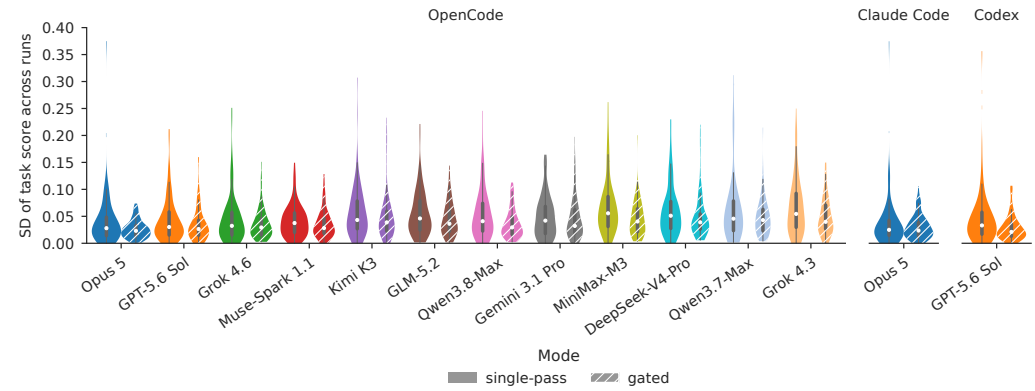

Figure A2: **Run-to-run consistency.** Each violin covers one model/harness combination in one mode and shows, over the 97 tasks, the distribution of the standard deviation of a task's score across its three runs. The inner bar marks the interquartile range and the white dot marks the median.

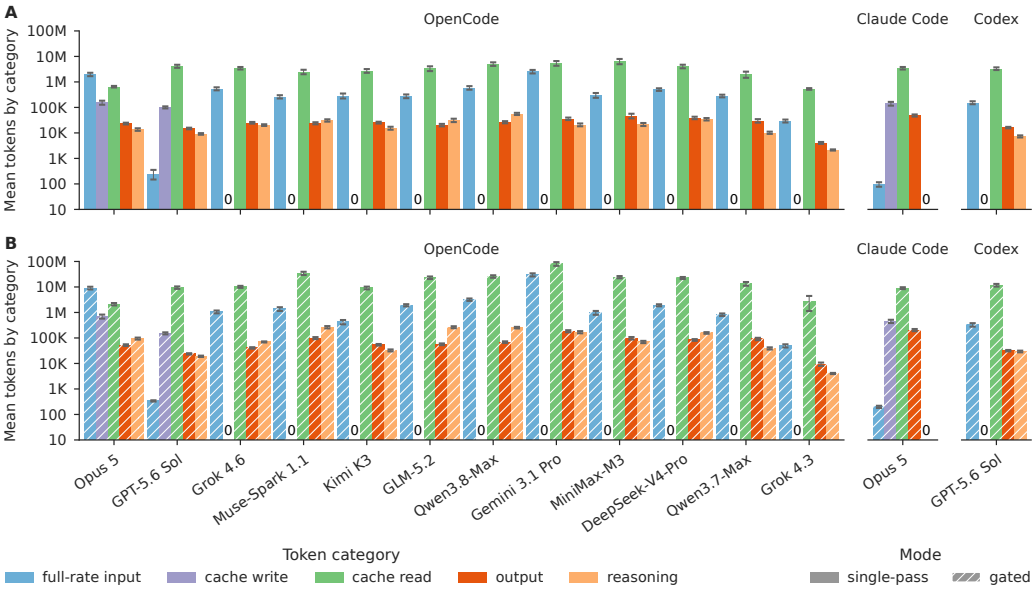

Figure A3: **Tokens per task by category.** (**A**) Single-pass mode. (**B**) Gated mode. Each bar is a combination's mean tokens per task in one category, on a logarithmic axis. Error bars are 95% confidence intervals.

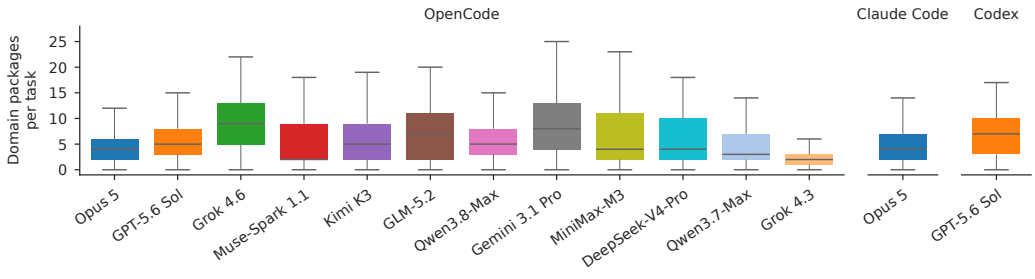

Figure A4: **Domain packages per task.** Each box covers one model/harness combination and shows, over the 97 tasks, the number of distinct domain packages the combination invoked on a task across its six runs, three in each mode.

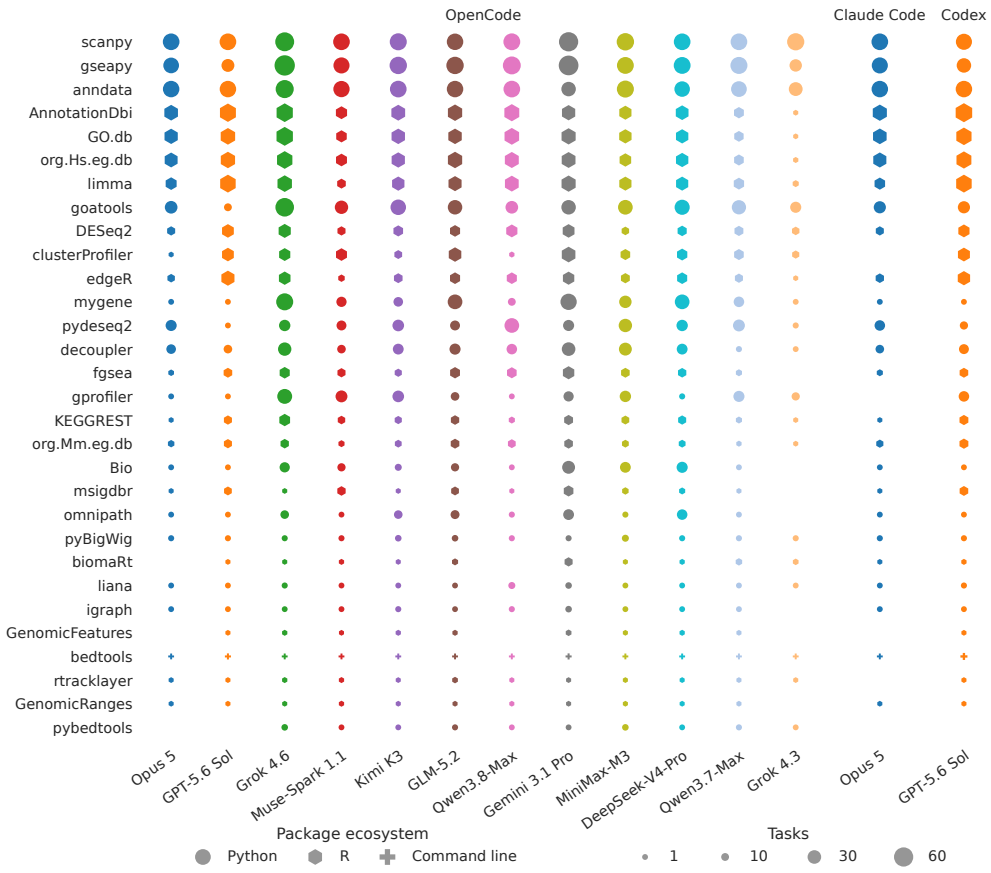

Figure A5: **Domain packages invoked by each model/harness combination.** Each row is one of the 30 most used domain packages, ordered by the number of tasks on which it was invoked, summed over all combinations (most used at the top). Marker area is the number of the 97 tasks on which that combination invoked the package across its six runs.

Table A1: **Pareto-optimality of models on OpenCode.** A check marks a model that is Pareto-optimal on a measure in one mode, meaning that no other model on OpenCode, the one harness on which all 12 models ran, scores higher at the same or lower value of that measure. N/A marks missing data.

Model	Cost		Time		Tool calls		Tokens	
	Single-pass	Gated	Single-pass	Gated	Single-pass	Gated	Single-pass	Gated
Opus 5	✓	✓	✓	✓	✓	✓	✓	✓
GPT-5.6 Sol	✓	✓	✓	✓				✓
Gemini 3.1 Pro								
GLM-5.2								
Grok 4.3	✓	✓	✓	✓	✓	✓	✓	✓
Grok 4.6	N/A	N/A		✓				✓
Kimi K3		✓						
MiniMax-M3	✓							
Muse-Spark 1.1	✓		✓		✓		✓	
Qwen3.7-Max							✓	
Qwen3.8-Max	N/A	N/A						
DeepSeek-V4-Pro	✓	✓						

B NOTATION AND PARAMETERS

We set $\tau = 0.7$ because the set and list graders give partial credit: a threshold of 1 would fail answers with a single extra or missing member, while a threshold at or below 0.5 would pass answers that are mostly wrong. Setting $\theta = 3$ bounds the cost of gated mode at 3 times that of single-pass mode and mirrors the review-and-revise cycle a scientist would tolerate before taking over. The time limit $t_{\max} = 1800$ s exceeds the typical turn time by a wide margin, so a timeout reflects unproductive work rather than a tight budget. Three runs per task ($k = 3$) is the smallest replicate design that lets us report run-to-run variance (Figure A2) while keeping the total at 8,148 sessions (97 tasks, 14 model/harness combinations, two modes, and three runs).

Table B1: **Notation.** Symbols used in the OmicsBench formulation.

Symbol	Meaning
$T = (P, D, G)$	A task, composed of a task prompt P, biological dataset D, and DAG G
P	The task prompt of a task
D	The biological dataset of a task
$G = (V, E)$	The task DAG, with nodes V and edges E
$v \in V$	A node of the task DAG, one subtask
$e = (u, v) \in E$	An edge of the task DAG, meaning that subtask v depends on the answer to subtask u
$v = (v_p, v_e)$	A subtask: its prompt v_p and its evaluation set v_e
v_p	Subtask prompt
$v_e = \{(g_1, y_1), (g_2, y_2), \ldots\}$	The graders of subtask v, each paired with its answer key
a_v	The agent's answer file for subtask v
$g_i(a_v, y_i) \in [0, 1]$	The score of grader g_i on its field in a_v
$s_v \in [0, 1]$	The score of subtask v, the mean of its grader scores, $\frac{1}{\lvert v_e \rvert} \sum_i g_i(a_v, y_i)$
S	The score of a task, the mean of its subtask scores, $\frac{1}{\lvert V \rvert} \sum_{v \in V} s_v$

Table B2: **Benchmark parameters.** OmicsBench settings and their values in this study.

Parameter	Symbol	Value
Gate threshold in gated mode	τ	0.7
Attempt limit per subtask in gated mode	θ	3
Time limit per turn (s)	$t_{\max}$	1800
Runs per task in each mode	k	3

C BENCHMARK PROMPTS

C.1 BENCHMARK SYSTEM PROMPT AND TASK PROMPT

```
You are a computational biologist. You have access to a computational
environment with standard bioinformatics tools.

Your workspace is organized as follows:
- /data/                -- input data files (read-only)
- /workspace/answers/   -- write ONLY your graded answer files here
(JSON)
- /workspace/scratch/   -- working area for intermediate files, scripts,
etc.

You will receive a series of analysis subtasks. For each subtask, you
will
be told what to do and where to write your answer. Complete each subtask
before moving on to the next.

How to succeed:

- Work autonomously. Do not ask for clarification. If an instruction is
  ambiguous, note your interpretation in your reasoning, pick the most
  reasonable one, and proceed.

- The graded artifact is the file you write at
  /workspace/answers/<requested_filename>, not anything you print in your
  response. If you describe the answer but do not write it to the correct
  file, you score zero. Writing the file is the job.

- Write exactly to the path and filename the subtask specifies. Do not
  rename, pluralize, change the extension, or route the file elsewhere.
  If a
  subtask asks for foo.json, the answer lives at
  /workspace/answers/foo.json.

- Answer files are strict JSON. No comments, no trailing commas, no
markdown
  fences, no wrapping the answer in extra keys. The file's top-level
  structure must match the schema in the subtask literally.

- For multiple-choice answers, write the selected answer exactly as one
of
  the choices shown in the subtask. Letter case is ignored by the grader,
  but
  spelling, punctuation, and qualifiers must match the listed choice.

- Before writing code that processes an input file, inspect it (shape,
  columns, dtypes, first few rows). Do not assume structure from memory.

- Commit to your best answer. Do not write placeholder values like "TBD",
  "uncertain", or null when the subtask asks for a concrete value.

- Do not delete files in /workspace/scratch/ between subtasks; later
```

```
  subtasks may depend on them.

Below, you will find a description of the data that you will be using to
accomplish your subtasks. Read through it carefully before starting any
analysis.

<task prompt>

Data modalities:
- <modality>

Organisms:
- <organism>

Files available in /data/:

1. <file name>
   - Description: <file description>
   - Format: <format>
   - Data modalities: <modality>
   - Organisms: <organism>
   - Platforms: <platform>
   - Path: /data/<file name>.<format>
```

C.2 SUBTASK PROMPT

```
Subtask: <subtask name>

<subtask instruction>

Write your answer as JSON to /workspace/answers/<subtask name>.json. The
schema below uses placeholders like ``<integer>``, ``<string>``,
``<number>``, ``<boolean>``, and ``<any>``. You MUST replace every
placeholder with an actual value of that type. Do NOT output the literal
placeholder string (``<integer>`` as a string value is never correct). The
shape around the placeholder(s) (brackets, braces, keys, and list
wrappers) is literal.

<answer schema>
```

In gated mode, a subtask whose predecessor took more than one attempt carries a note after the subtask line, naming the workspace it inherits:

```
Note: Your workspace has been restored to the state from attempt <n> of
the previous subtask (<previous subtask name>), which was your best
attempt. Files on disk may differ from your most recent attempt.
```

When the runner includes the numeric score in its feedback (Appendix C.3), the note reads instead:

```
Note: Your workspace has been restored to the state from attempt <n> of
the previous subtask (<previous subtask name>), which had the best score
(<score>). Files on disk may differ from your most recent attempt.
```

When every attempt at the predecessor failed, the note reads instead:

```
Note: Every attempt at the previous subtask (<previous subtask name>)
failed. The workspace may be in an inconsistent state; verify file
contents before relying on them.
```

A subtask whose answer carries group labels appends this block:

```
Grouped fields:
Use the listed group labels exactly for each grouped answer field.

- "<field name>": ["<group label>", ...]
```

A subtask whose answer carries a closed set of choices appends this one instead:

```
Choice fields:
Multiple-choice fields are scalar values. Multi-select fields are lists.
For list-valued fields, include every correct choice and no unsupported
choices. Answer exactly as one of the listed choices. Letter case is
ignored by the grader for string choices, but spelling, punctuation, and
qualifiers must match the listed choice.

- "<field name>": ["<choice>", ...]
```

A subtask whose answer carries a labeled numeric matrix appends this one:

```
Labeled matrix fields:
Use these row and column labels exactly for each labeled numeric matrix
field.

- "<field name>" rows: ["<row label>", ...]
- "<field name>" columns: ["<column label>", ...]
```

C.3 Retry prompt

Below is the default retry prompt.

```
Your previous attempt scored below the required threshold. Please review
your approach and try again. Write your updated answer to
/workspace/answers/<subtask name>.json.
```

An attempt that cannot be graded at all is returned with the reason instead:

```
Your previous attempt could not be scored. Reason: <reason>

Please fix the issue and write a valid answer to
/workspace/answers/<subtask name>.json.
```

The runner can also include the numeric score in its feedback. In that form, the retry prompt reads as follows, the unscorable form is unchanged, and the workspace-restore note in Appendix C.2 gives the kept attempt's score.

```
Your previous attempt scored <score> out of 1.0, which is below the
required threshold. Please review your approach and try again. Write your
updated answer to /workspace/answers/<subtask name>.json.
```

D Example task

Example task prompt

> The adult human kidney cortex was profiled with paired single-nucleus RNA sequencing (transcriptome) and single-nucleus ATAC sequencing (chromatin accessibility, provided as a per-cell gene-activity matrix plus raw fragments). Both single-nucleus datasets are annotated with cell-type and cell-state labels, including a rare VCAM1-expressing proximal tubule subpopulation (labeled PT_VCAM1) that sits alongside the remaining, differentiated proximal tubule. The scientific goal is to reconstruct the molecular identity and regulation of this VCAM1+ proximal tubule state from the data: define its transcriptional signature relative to the remaining proximal tubule, determine how the genes that drive its emergence change and whether those changes are mirrored at the chromatin-accessibility level, determine which transcription-factor programs switch as the state appears, and characterize the biological programs it activates. Use the annotated proximal tubule cells in the provided matrices to derive these results.

Table D1: **Files in the example task's dataset.**

File	Description
`snrna_expression.h5ad`	Single-nucleus RNA-seq of adult human kidney cortex: log-normalized expression matrix (cells × genes) with per-cell cell-type/cell-state annotations in `obs` (`author_cell_type` includes PT for the remaining proximal tubule and PT_VCAM1 for the VCAM1+ proximal tubule subpopulation, plus other nephron cell types). Gene symbols are in `var['feature_name']`.
`snatac_gene_activity.h5ad`	Single-nucleus ATAC-seq of adult human kidney cortex represented as a per-cell gene-activity matrix (log-scaled accessibility aggregated over each gene's body and promoter; cells × genes) with per-cell cell-type/cell-state annotations in `obs` (`author_cell_type` includes PCT and PST for proximal tubule segments and PT_VCAM1 for the VCAM1+ proximal tubule subpopulation, plus other nephron cell types). Gene symbols are in `var['feature_name']`.
`snatac_fragments.tsv.gz`	Raw single-nucleus ATAC-seq fragments for the same kidney cortex nuclei, block-compressed TSV keyed by genomic coordinate and cell barcode. Needed for peak- or motif-level analyses such as transcription-factor motif activity; the gene-activity matrix suffices for gene-level accessibility comparisons.
`snatac_fragments.tsv.gz.tbi`	Tabix index for the single-nucleus ATAC-seq fragments file, enabling coordinate-range queries.

Table D2: **Subtask prompts of the example task.** Numbers on the left refer to the boxes in the flowchart of Figure 1A, which omits `pt_vcam1_programs`.

	Subtask	Depends on	Subtask prompt
❶	`pt_vcam1_signature`	Root subtask, no upstream subtasks	Using the single-nucleus RNA-seq expression matrix, contrast the VCAM1+ proximal tubule subpopulation (PT_VCAM1) against the remaining proximal tubule (PT) and derive the differential transcriptional signature of this cell state, testing genes detected in at least 20% of cells in either group with a log fold-change threshold of 0.25 and Bonferroni-adjusted $p < 0.05$. Report the 15 most strongly up-regulated and, separately, the 15 most strongly down-regulated genes in PT_VCAM1, ranked by average log fold-change as computed by Seurat's FindMarkers (avg_logFC: the natural log of one plus the mean of expm1(expression) in PT_VCAM1, minus the same quantity in PT, on the provided log-normalized values). Provide two lists of gene symbols keyed as 'upregulated' and 'downregulated'.
❷	`pt_vcam1_transition_drivers`	`pt_vcam1_signature`	Order the proximal tubule cells along a transition from the differentiated proximal tubule (PT) to the VCAM1+ state (PT_VCAM1). The source study reports four hallmark genes whose expression changes define this transition: VCAM1, TPM1, SLC5A12, and SLC4A4. Determine the direction in which each of them changes along the transition and report them grouped by direction: genes induced (rising toward PT_VCAM1) under group label 'induced', and genes repressed (falling toward PT_VCAM1) under group label 'repressed'. Each group holds gene symbols.

	Subtask	Depends on	Subtask prompt
3	pt_vcam1_regulators	pt_vcam1_signature	Nominate the transcription-factor programs that switch as the proximal tubule adopts the VCAM1+ state by computing transcription-factor motif activity from the single-nucleus ATAC fragments (for example chromVAR deviations over JASPAR vertebrate motifs), comparing PT_VCAM1 nuclei with the differentiated proximal tubule nuclei (the PCT and PST segments). Identify the transcription factor whose motif activity is most clearly lost in PT_VCAM1 and the two transcription factors whose motif activity is most strongly gained. You may corroborate the direction using the PT_VCAM1 signature you derived, transcription-factor gene expression, and gene activity. Where motif activity alone cannot separate paralogs that share a near-identical motif, report the paralog whose own expression and gene activity move in the same direction as the motif activity. Report each regulator by its own gene symbol, grouped as 'repressed_regulators' (lost in PT_VCAM1) and 'induced_regulators' (gained in PT_VCAM1).
	pt_vcam1_programs	pt_vcam1_signature	Run a functional-enrichment analysis on the full PT_VCAM1 signature, up- and down-regulated genes together, and select the biological programs that are enriched in this subpopulation relative to the remaining proximal tubule. Choose all applicable programs from the provided options.
4	pt_vcam1_accessibility_concordance	pt_vcam1_transition_drivers	Using the single-nucleus ATAC gene-activity matrix, test whether the transition-driver genes identified for the PT-to-PT_VCAM1 transition show concordant changes in chromatin accessibility. Compare gene activity in PT_VCAM1 cells against the differentiated proximal tubule cells (the PCT and PST segments) and report which of those driver genes show increased accessibility in PT_VCAM1 under group label 'increased_accessibility' and which show decreased accessibility under group label 'decreased_accessibility'. Each group holds gene symbols.

E CODE AND DATA AVAILABILITY

OmicsBench's code and public data are available online at https://huggingface.co/datasets/anonymous3489/OmicsBench-supplementary-data. The dataset holds the OmicsBench runner, comprising the evaluation harness and every grader, under an MIT license, three of the 97 tasks with their datasets, among them the example task shown in Figure 1 and Appendix D, and the identifier databases against which the graders match. The three public tasks, including their answer keys, are released under a CC BY 4.0 license, and their datasets under the licenses of the studies that released them. The remaining tasks are held back as a private test set to keep the benchmark useful for evaluating future models, and researchers who wish to evaluate an agent on these tasks may contact the authors.

F GRADER CATALOG

Table F1: **OmicsBench grader catalog.**

Grader	Description
absolute_numeric_tolerance	Compares one numeric value using an absolute error band.
abundance_table	Scores an unordered table of entity names paired with numeric abundances or quantities.
amino_acid_set	Scores an unordered set of amino acids using one-letter, three-letter, and full-name normalization.

Grader	Description
amr_gene_set	Scores an unordered set of antimicrobial resistance genes using CARD ARO synonym resolution.
case_insensitive_string	Compares one scalar string after lowercasing both sides.
case_insensitive_string_list	Scores an unordered set of literal strings while ignoring case.
category_count_table	Grades categorical metadata summaries such as cell-type counts, tissue compositions, sample-group sizes, or condition breakdowns.
coordinate_point_set	Scores unordered 2D or 3D point records by Euclidean distance, optionally requiring a point identifier to match before distance is scored.
curve_fit_params	Scores a four-parameter response curve fit using parameter-specific numeric tolerances.
differential_feature_set	Scores unordered differential-result rows keyed by feature with effect size and significance statistics.
distribution	Compares keyed numeric distributions after exact key alignment.
docking_pose_set	Scores unordered docking-pose records with ligand identity, docking score, and a nested set of interacting residues.
enrichment_result_set	Scores unordered enrichment-result rows keyed by term with significance and optional count statistics.
entity_pair_set	Scores unordered directed pair records with exact string endpoints.
exact_string	Compares one scalar string with no normalization.
exact_string_list	Scores an unordered set of case-sensitive literal strings.
gene_pair_set	Scores unordered directed gene-gene pair records.
gene_set	Scores an unordered set of gene names using NCBI Gene synonym resolution for the configured organism.
gene_taxon_link_set	Scores unordered records linking an exact gene identifier to a taxon.
grouped_entity_set	Scores records where each group label owns an unordered set of entity or label items.
grouped_ranked_list	Scores records where each group label owns an ordered list of entity or label items.
interval	Scores one genomic interval by exact chromosome match and reciprocal span overlap.
interval_pair_set	Scores an unordered set of paired genomic loci.
interval_set	Scores an unordered set of genomic intervals.
labeled_numeric_matrix	Scores a two-axis numeric table represented as unordered matrix cells.
labeling	Compares two parallel label assignments as relabel-invariant clusters.
metabolite_set	Scores an unordered set of metabolite names using HMDB synonym resolution.
mirna_set	Scores an unordered set of miRNA names using miRBase normalization.
multiple_choice	Scores one scalar answer against a fixed homogeneous set of allowed choices.
multi_select_choice	Scores a list of selected choices against a fixed homogeneous set of allowed choices.
numeric_list	Scores an ordered numeric vector either by per-position tolerance or by correlation.
numeric_list_mean	Scores a fixed-order numeric vector by averaging per-position absolute tolerance scores.
numeric_tolerance	Compares one numeric value using absolute, relative, or log-scale tolerance.
pair_set	Scores unordered records with two string/entity endpoints named “a” and “b”.
protein_site_set	Scores unordered records that identify a protein/gene and residue site, optionally with a semantic site label.
ranked_amino_acid_list	Scores an ordered amino-acid list after one-letter, three-letter, and full-name normalization.
ranked_amr_list	Scores an ordered list of AMR genes after CARD normalization.
ranked_gene_list	Scores an ordered gene list after gene synonym resolution.

Grader	Description
`ranked_list`	Scores an ordered list of strings after resolving each element with a selected string matcher.
`ranked_metabolite_list`	Scores an ordered list of metabolites after HMDB normalization.
`ranked_mirna_list`	Scores an ordered miRNA list after miRBase normalization.
`ranked_protein_site_list`	Scores ordered protein+residue-site records, optionally including a case-insensitive site label.
`ranked_residue_site_list`	Scores an ordered residue-site list after amino-acid and PTM-prefix normalization.
`ranked_taxa_list`	Scores an ordered list of taxa after NCBI Taxonomy normalization.
`record_set`	Grades a variable-length list of objects using a per-field schema.
`relative_numeric_tolerance`	Compares one numeric value using error relative to the expected value.
`residue_site_set`	Scores an unordered set of residue-site strings after amino-acid and PTM-prefix normalization.
`scored_entity_set`	Scores unordered entity-score rows where the entity identity gates the row and a single numeric score receives fractional credit.
`scored_interval_set`	Scores an unordered set of genomic intervals, each carrying a numeric evidence field.
`spectral_peak_set`	Scores unordered spectral peak records with assignment, position, and intensity.
`string_match`	Compares one scalar string using a selected literal, biological entity, amino-acid, or residue-site matching mode.
`string_set`	Scores an unordered set of strings using a selected literal or semantic string matcher.
`struct_set`	Scores unordered records made only of named string/entity fields.
`taxa_set`	Scores an unordered set of taxa using NCBI Taxonomy synonym resolution.
`taxon_pair_set`	Scores unordered directed pair records whose endpoints are taxa.
`variant_call`	Scores one small variant call with chromosome, position, reference allele, and alternate allele fields.
`variant_set`	Scores an unordered set of small variant calls.
`weighted_entity_pair_set`	Grades a set of network, interaction, regulatory, or association edges where each edge has two named endpoints (a, b) and a numeric weight, confidence, correlation, or effect size.

G IDENTIFIER DATABASES

Table G1: **Identifier databases that back OmicsBench's synonym-aware graders.**

Database	Description	Reference
NCBI Gene	Matches between different gene nomenclature systems.	Brown et al. 2015
NCBI Taxonomy	Matches between different organism naming systems.	Schoch et al. 2020
HMDB	Matches between different metabolite naming systems.	Wishart et al. 2022
miRBase	Matches between different miRNA naming systems.	Kozomara et al. 2019
CARD	Matches between different antimicrobial resistance gene naming systems.	Alcock et al. 2023

H A PRIORI FAILURE MODE ANNOTATION

For each failed attempt, the annotating LLM judge received the subtask prompt, the grader configuration with the answer key, the submitted answer, the score and its sub-scores, and the agent's transcript for that turn, with model and harness names removed. It could assign at most five failure modes to an attempt, each backed by a verbatim quote from the transcript.

To measure the LLM judge's reproducibility, we annotated a separate validation set of 100 failed attempts twice with the Opus 5 LLM judge, shuffling the attempts into different batches each time.

With each pairing of an attempt with one of the 13 failure modes treated as a yes-or-no decision on whether the mode was assigned, the two passes agreed with a Cohen's kappa of 0.76 over those decisions.

Table H1: **A priori failure modes used to annotate failed attempts.**

	Failure mode	Definition
	Environment setup failure	The agent could not get required tools, packages, references, or paths into a usable state, and the unresolved setup problem prevented or materially blocked the intended analysis.
	Input misinterpretation	The agent misread the staged inputs, confused sample identities, or applied an analysis intended for a different data type. Applies only when the data themselves were misread; a later analysis choice that diverged is Method misconfigured.
	Wrong method substitution	The agent used a method materially different from the one the reference analysis requires, such as a different tool family, resource, or statistical approach, including hand-built stand-ins (keyword matching, reimplemented pipelines) for a standard resource it did not use.
	Method misconfigured	The agent used a suitable method but chose parameters, contrasts, references, normalizations, or filters that materially diverge from the reference analysis. Not added when a more specific mode captures the same choice.
	Incomplete data	The agent analyzed fewer samples, conditions, or features than the subtask specified, so the analysis was incomplete even where the reduced analysis ran.
	Output format violation	The answer did not match the subtask's answer contract: wrong filename or location, identifier namespace, naming, units, granularity, key names, or list length.
	Synthetic or placeholder output	The agent wrote synthetic, dummy, or memory-derived values instead of deriving the output from the staged data, including hand-assembled canonical panels substituted for its own computed result, whether openly or to conceal an incomplete analysis.
	Premature termination	The agent stopped before writing a scorable answer because it exhausted a turn or time budget, or declared completion before the work was finished. A timed-out attempt with no answer file is this mode even when no transcript exists.
	Inefficient analysis	The agent spent its time budget unproductively, on a materially large detour into work the subtask did not ask for or on re-running completed steps, and the lost time plausibly cost points.
	Repetitive retry loop	The agent repeatedly retried substantially the same failing command or approach without a meaningful diagnosis or change of strategy.
	Answer set too broad	The subtask asked for a compact, discriminating set; the agent submitted everything clearing a permissive cut or a fixed top-N block of its ranking, so co-ranked extras diluted the set's precision.
	Answer set too strict	The agent stacked stringent or conjunctive cutoffs, agreement demands, or exclusion lists until true members of the expected set were removed before submission.
	Selection criterion mismatch	The agent selected or ranked candidates by a measure other than the one the subtask stated, so entities satisfying the wrong criterion filled the answer. Distinguished from Method misconfigured by being about which quantity ranked the candidates, not how it was computed.

I UNSUPERVISED FAILURE MODE DISCOVERY

The failure descriptions were embedded with `openai/text-embedding-3-large`, reduced to five dimensions with UMAP (cosine distance, 15 neighbors, minimum distance 0), and clustered

with HDBSCAN using leaf cluster selection and a minimum cluster size of 60, which gave 15 clusters holding 54% of failed attempts. The naming LLM judge saw each cluster's eight most central descriptions, with three descriptions from its boundary for contrast. We set three of these clusters aside because they captured no failure of the agents. Two captured tool outages (221 and 158 attempts), and the third merged attempts that used the whole time budget without writing an answer with attempts from a single task that took gene coordinates from the wrong genome version (137 attempts).

Table I1: **Emergent failure modes recovered by unsupervised discovery.** The modes are grouped into five families.

	Failure mode	Definition
Too much submitted		
1	Same number of top genes taken from every group, so lists were mostly extras	The agent took the same fixed number of top-ranked genes from every group, even though the subtask asked for the few genes that set each group apart, so most of what it submitted was extras, and the score fell.
2	Every gene passing its thresholds submitted instead of a short list	The agent submitted every gene that passed its thresholds instead of selecting the short list the subtask asked for, so the submission was mostly extras.
3	Ranked by test score only, so ubiquitous genes displaced the markers	The agent ranked genes by the test score alone with no expression filter, which let genes expressed everywhere take the top places, so the specific markers the subtask asked for were pushed off the list.
4	Genes picked by keyword match instead of a statistical test	The agent chose genes by matching keywords in their names or annotations rather than by the statistical test the subtask required, which pulled in unrelated genes that matched the words and left out significant genes that did not.
Too little submitted		
5	Two datasets each over-filtered before intersecting, so almost nothing remained	The agent filtered each of two datasets strictly on its own before intersecting them, when the subtask defined the answer as the intersection at a stated cutoff, so almost nothing survived, and most of the expected set was missing.
The agent's own criterion in place of the subtask's		
6	Extra cutoffs added on top of the subtask's criterion, dropping real genes and admitting unrelated ones	The agent added extra cutoffs on fold change, detection, and significance on top of the single criterion the subtask used, which dropped expected genes that missed any one cut while unrelated genes that happened to pass them all filled the list, so the answer both missed real members and carried extras.
7	Only a self-written candidate list tested, so genes outside it were never found	The agent tested only a list of candidate genes it wrote itself rather than the whole dataset, so any expected gene missing from its list could never be found.
8	Right category, but its example genes picked by a shortcut	The agent identified the right category but chose its example genes by keyword matching or its own ranking instead of taking them from the analysis behind the category, so the example list barely overlapped the expected one.
Prior knowledge in place of the data		
9	Marker genes from memory added to the computed list, counted as extras	The agent added well-known marker genes from memory to the list it had computed, which put genes the dataset did not support into the answer, so they counted as extras.
10	Answer written from memory without opening the data	The agent wrote the answer from memory or from earlier turns without opening the data, which produced the generic textbook answer where the subtask asked for the result specific to this dataset, so members were wrong or swapped.

	Failure mode	Definition
A different procedure substituted		
11	Uncorrected p-values or a permissive test, so too many genes called significant	The agent used uncorrected p-values or the more permissive version of a test when the expected count came from the corrected test, causing it to report several times too many significant genes.
12	Time-course patterns assigned by hand-written rules instead of clustering	The agent sorted genes by the pattern of their expression over time using rules it wrote by hand, such as "rises then falls", instead of clustering the measured curves, so genes whose curves did not fit its rules landed in the wrong group.

J Model and harness configurations

Table J1: **Models and harnesses evaluated in this study.** Reasoning effort was set to high for every model except MiniMax-M3 and Qwen3.7-Max, for which OpenRouter exposes no effort setting. The harness versions were OpenCode 1.18.11, Claude Code 2.1.222, and Codex 0.142.5.

Model	Harnesses
Opus 5	OpenCode, Claude Code
GPT-5.6 Sol	OpenCode, Codex
Gemini 3.1 Pro	OpenCode
GLM-5.2	OpenCode
Grok 4.3	OpenCode
Grok 4.6	OpenCode
Kimi K3	OpenCode
MiniMax-M3	OpenCode
Muse-Spark 1.1	OpenCode
Qwen3.7-Max	OpenCode
Qwen3.8-Max	OpenCode
DeepSeek-V4-Pro	OpenCode

K Runtime environment details

Every run took place in its own Docker container, built from one standard image shared by all model/harness combinations. The container ran Debian 13 (Trixie), and the host machine ran Ubuntu 24.04 LTS (Noble Numbat). Each container was given four processor cores, 12 GB of memory, and 20 GB of storage, with the task dataset mounted read-only. Containers had no network access, so an agent could use only what the image and the task dataset provided. The Dockerfile that defines the benchmark runtime is available online (Appendix E). Within an OmicsBench session, the time limit per turn ($t_{\max}$) was 1800 seconds, and in gated mode the gate threshold τ was set at 0.7 and the attempt limit θ at 3 (Table B2).

L Judge prompts

Every judge call below carries the same system prompt:

```
You are an expert reviewer of computational biology work. You read
transcripts of automated agents doing omics analyses inside a container
and you describe what they did with precision and without judgement
words. You answer with a single JSON object and nothing else: no prose
before or after it, no code fences.
```

L.1 TRAJECTORY NARRATIVE AND A PRIORI FAILURE MODE ANNOTATION

```
You are annotating attempts from a benchmark of agents doing real omics
analyses. Each agent worked in a container with the data under /data/ and
had to write an answer file under /workspace/answers/. A grader then
scored the answer file against a hidden key (score 0 to 1). The agent
never saw the key.

Below are <batch size> evidence bundles, one attempt each, from different
questions. Each bundle shows: the node prompt the agent received, the
grader configuration with the expected value, the attempt record with the
submitted answer, the score and its sub-scores, and the agent's own turn
of the transcript (its messages, tool calls and tool results, truncated
where long). Model names and harness names have been removed; the tool
syntax may differ between bundles and that is not something to classify
on. A bundle marked EVIDENCE MISSING has no transcript (the attempt timed
out); it shows workspace file timestamps and the agent log instead.

Your job has two parts per bundle. Work only from the evidence in that
bundle.

PART 1, for EVERY bundle: an approach-and-outcome narrative, two to three
sentences on what the agent tried and what happened with it. Sentence one
is the route taken: which files it opened, how it normalized or filtered,
which statistical method or tool, what checks it ran. The remaining
sentence or two give the mechanism of the result: how the specific
choices led to the specific answer that was submitted, compared with what
the grader expected. Write it the same way whether the score is high or
low: describe, do not judge. DO NOT use verdict vocabulary anywhere in
the narrative: no "succeeded", "failed", "failure", "correct",
"incorrect", "wrong", "right", "mistake", "error", "bad", "good",
"properly", "improperly". Name the route and the consequence instead
("used the raw intensity columns rather than the normalized ones, so the
enriched set was dominated by contaminant proteins"). Also give the step
index in the transcript where the route diverged from what the grader
needed, or null if the score is at or above the threshold or you cannot
tell.

ALSO for every bundle, a ROUTE field: ONE sentence describing the
analytical route in fully task-agnostic terms. Abstract away every
task-specific entity: no gene, protein, or pathway names, no tissue,
disease, organism, or cell-type words, no file names, no database names
beyond their kind (say "the annotation database", "the expression
matrix", "the target entities", "the two groups"). Keep only the shape of
the work: what kind of data was loaded, how it was transformed or
normalized, what was computed or tested, how candidates were selected or
ranked, what was checked, and what kind of thing was submitted. Example:
"loaded the expression matrix, normalized by library size, ran a
two-group differential test, kept everything passing a significance cut,
and submitted the full list without pruning." This field is used to group
attempts by route across different tasks, so any task-identifying word
defeats its purpose.
```

```
PART 2, ONLY for bundles marked SCORE BELOW THRESHOLD: failure-mode tags
from the fixed vocabulary below. Assign between zero and 5 tags: every
mode whose mechanism the evidence concretely shows contributed to the
score falling below 0.7. Require concrete evidence for every tag; when no
mode applies, return an empty tag list (that is a valid and informative
answer, never force a tag). For each tag give: the mode name copied
verbatim; one or two sentences saying how this mechanism shows in this
attempt; a verbatim quote of at most 200 characters from the transcript
or the submitted answer showing it; and your confidence as "high" (the
transcript shows the mechanism directly), "medium" (the best reading of
partial evidence) or "low" (a plausible reading among several). If the
bundle has no transcript and the evidence cannot support any judgement,
use the single tag "undeterminable from evidence" with confidence "low"
and say in the narrative's last sentence what evidence is missing.

The vocabulary. These are the only mode names PART 2 may use:

<the a priori failure modes, each as "- <name>: <definition>">
- undeterminable from evidence: The bundle does not contain enough
evidence to support any judgement; used alone, only for bundles without a
transcript.

Rules that override everything else:
- Tag the agent's work, never the harness, the tool syntax, or the
container.
- A tag needs concrete evidence in this bundle; no tag when none applies.
- Tags name mechanisms that lost points, not incidental oddities. An
attempt can carry several tags when several mechanisms each show
concretely.
- Use only evidence inside the bundle. Do not use your own domain
knowledge to decide what the key "should" be beyond what the grader
configuration shows.
- Every quote must appear verbatim in the bundle. Never quote the
grader's expected value.

Answer with ONE JSON object of exactly this shape, and nothing else:

{
  "bundles": [
    {
      "key": "<bundle key>",
      "narrative": "<two to three sentences>",
      "route": "<one task-agnostic sentence>",
      "diverged_at_step": <integer or null>,
      "tags": [
        {
          "mode": "<mode name from the vocabulary>",
          "mechanism": "<one or two sentences>",
          "evidence_quote": "<verbatim, at most 200 characters>",
          "confidence": "<high | medium | low>"
        }
      ]
    }
  ]
}

For a bundle whose score is at or above the threshold, set "tags" to
null. For a below-threshold bundle where no mode applies, set "tags" to
[]. Bundle keys in this batch: <bundle keys>.

<evidence bundles>
```

L.2 TASK-AGNOSTIC FAILURE DESCRIPTIONS

```
You will read <batch size> short narratives. Each describes how an
automated agent's analysis of an omics dataset led to the answer it
submitted, and how that answer related to what the grader expected. For
EACH narrative write ONE sentence, in fully task-agnostic terms,
describing the MECHANISM by which the submitted answer diverged from what
was expected.

Rules for the sentence:
- Abstract away every task-specific entity: no gene, protein, pathway,
cell-type, tissue, disease, organism, chromosome or sample names, no
file, column or database names, no task-specific numbers. Say "the target
features", "the two groups", "the annotation column", "the expected set",
"the reference group".
- Keep the SHAPE of what happened: which decision or step produced the
divergence (a selection rule, a threshold, a reference group, a
normalisation, a label format, a data subset, a stopping point, a
fabricated value), and how the submitted answer differed from the
expected one in kind (more items, fewer items, different items, wrong
format, no items, partially matching).
- Be specific about the mechanism, not the category. Do NOT write
category labels such as "answer set too broad", "method misconfigured",
"wrong method", "incomplete data", "premature termination", "placeholder
output", "format violation"; write what the agent did instead ("kept
every feature passing a nominal cut and submitted the whole list where a
compact discriminating set was expected").
- Describe, do not judge: no "wrong", "incorrect", "failed", "error",
"mistake", "properly".
- Write the same way for every score level. If the narrative says nothing
about the mechanism (for example only that time ran out), say exactly
that in task-agnostic terms.
```

L.3 CLUSTER NAMING

```
You are naming one cluster of sentences. Each sentence describes, in
task-agnostic terms, the mechanism by which an automated agent's
submitted answer on an omics analysis task diverged from what was
expected. The sentences were grouped by embedding similarity, so they
should share one mechanism. You are given the <count> sentences closest
to the cluster centre and, for contrast, sentences near the cluster
boundary; the name should fit the former better than the latter.

Rules:
- The name is 3 to 8 words naming the MECHANISM as a bioinformatician
would describe a colleague's choice: what the agent did and, if it fits,
how the answer differed in kind (e.g. "fixed top-N block, extras dilute
the set", "candidate panel chosen by hand, misses items outside it",
"answer written from memory, data never opened").
- Name the mechanism, not a judgement and not a category: no "failed",
"wrong", "incorrect", "error", "bad"; do not reuse taxonomy labels such
as "answer set too broad" or "method misconfigured".
- Use only what the sentences show.
- If the sentences do not share one mechanism, set "coherent" to false
and name the largest coherent subset.
```