AGENTS ARE SYSTEMS, NOT MODELS: RETHINKING AGENTIC EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent evaluations increasingly go beyond a single success rate, reporting metrics such as cost, consistency, and robustness. Yet they typically treat the agent itself as fixed. In practice, an agent is a configurable system: users decide what to tell it, how long to let it run, and which model to use, and each of these choices can change how well and how consistently it performs. We study these choices on a new benchmark of four scientific tasks, where a coding agent must find and correctly operate a published specialist model. We investigate five parts of the agent's configuration: task information, reasoning, self-verification, time budget, and backbone model. We find substantial run-to-run variability, with approximately 54% of the outcome variance coming from repeating the same configuration rather than changing it. Across configurations, the information provided to the agent has the largest effect, exceeding both time budget and model size, while also reducing cost and improving calibration. Configuration choices also interact: additional time helps only when the agent has sufficient information or a capable enough model to use it. Finally, a trajectory-based taxonomy of agent behavior reveals that prompting an agent to verify its answer has little effect on its verification behavior, whereas providing a dedicated verification tool changes that behavior substantially. These results suggest that agents should be evaluated as configurable systems themselves, and that some desired behaviors are more effectively implemented in the system than requested through prompting. We will release the benchmark and more than 18,000 agent trajectories.

1 INTRODUCTION

When agents are used to solve complex tasks, we care not only about whether they succeed, but also about how reliably they do so, and about which parts of their configuration determine capability and reliability. Recent work broadens what is measured about an agent: Kapoor et al. (2025) argue that evaluations should not focus only on performance, but also account for cost and comparability between methods. Rabanser et al. (2026) show that capability gains do not automatically translate into reliability gains, and argue for a decomposition of agent reliability into consistency, robustness, predictability, and safety. It remains less clear how an agent's configuration itself shapes these measurements: which of its parts drive capability and reliability, and how these parts interact.

Scientific workflows are a natural setting to study this. Many scientific tasks rely on specialist models (trained for a specific scientific domain or task) that a general-purpose LLM cannot directly replace, so an agent cannot solve them from its own knowledge. Instead, it has to find the right model, prepare the input data, and write the code to run it, and each of these steps can be affected by how the agent is configured. Published results of these specialist models further provide reference scores against which the agent's outcome can be checked. One example is estimating redshift from galaxy images. Fine-tuning a multimodal LLM to process galaxy images would be costly, but specialist models for this task already exist, and an agent can use them instead. We are interested in *how reliably agents can use such specialist models* in scientific workflows, and *what contributes to their effective use*.

In this paper, we use the following terms: the *model* (or *backbone*) is the LLM itself, with no memory across calls and no ability to act on its own. An *agent* wraps a model in a loop that lets it act repeatedly: read the current state, decide on an action, execute it, observe the result, and

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.

decide again. The agent's *harness* is the infrastructure that runs this loop: it dispatches tool calls and manages the model's context at each step, including memory and compression. Fundamentally, an agent is a *system* consisting of a model and a harness (de Macedo, 2026; Paniego & Roy Gosthipaty, 2026), which is reflected in recent implementations of production coding agents such as Codex or Claude Code (Lopopolo, 2026; Young, 2025; Shi et al., 2026).

We systematically evaluate how an agent's configuration affects how well it uses specialist models to solve scientific tasks. Concretely, we study five axes of the agent's configuration: (1) what *information* the agent is given; (2) whether the agent generates *reasoning* and keeps it across steps; (3) how strongly the agent is asked to *verify* its own answer; (4) how much runtime *budget* the agent has to solve the task; and (5) how strong the agent's backbone *model* is. By varying these axes on a new benchmark of four scientific tasks from astrophysics and genomics, we measure how each one contributes to the agent's outcome and reliability, and how they interact. In each task, the agent's outcome is compared against the specialist model on an evaluation from its original paper.

We find that agent outcomes are noisy: about 54% of the score variance among genuine attempts is run-to-run noise rather than an effect of the configuration. In our main experiments, the information given to the agent has the largest effect among the axes, outranking both the budget and the backbone model. The effect of one axis can also depend on another: additional budget mainly improves the score when the agent already has enough information to use it productively. Finally, an analysis of the agent's behavior shows that some behaviors, such as self-verification, are hard for the user to influence directly through prompting and should instead be shaped by the system's design.

In summary, our contributions are as follows: **(1)** We systematically study how the axes of an agent's configuration influence its outcome and reliability, finding that information outranks model and budget. **(2)** We create an AI-for-Science (AI4S) benchmark in which the agent has to use specialist models under controlled conditions. We release the benchmark and 18,240 agent trajectories with their produced artifacts, enabling reproducibility and further analysis of agent behavior. **(3)** We derive a taxonomy of agent behavior from these trajectories and use it to show that some behaviors are hard for the user to influence directly and should instead be shaped by the system's design.

2 Related Work

AI for Science. Recent capability gains in machine learning methods and foundation models enable a broad range of applications in scientific fields outside classical computer science (Jumper et al., 2021; Wang et al., 2023). Furthermore, the transition from isolated models to agents has intensified the interest in AI as a scientific companion or as a scientist itself (Wei et al., 2025; Gridach et al., 2025; Xin et al., 2025; Lu et al., 2026). We organize the field into two focus areas: *(re-)implementation* (Starace et al., 2025) and *discovery* (Majumder et al., 2025). Works on (re-)implementation reuse scientific publications or their codebase, and test how well agents can reproduce the findings, methods, or implementations (Wang et al., 2026; Chen et al., 2025; Kim et al., 2026; Bogin et al., 2024; Falck et al., 2026; Shen et al., 2026). In parallel, a growing set of works evaluates agents' ability to produce genuinely novel discoveries (Kirgis et al., 2026; Artiles et al., 2026; Si et al., 2025; Zahavy, 2026; Si et al., 2026; Messeri & Crockett, 2024). Our work is closer to the first area, i.e., (re-)implementation of existing work. We propose a coding-focused benchmark that enables the agent to freely explore, test, and write code to operate given specialist models, while its outcome is judged against their published performance. This makes the environment controlled and repeatable, so that the agent's configuration can be varied systematically and outcomes can be scored without manual verification.

Agents and Their Evaluation. As agents (Wooldridge & Jennings, 1995; Yao et al., 2023) become more capable, evaluating their performance (Chang et al., 2024) becomes more challenging. Their ability to tackle complex, long-horizon tasks (Kwa et al., 2025), potentially through multiple instances (Cemri et al., 2026), makes failures and capabilities harder to accurately estimate and understand, which makes a systematic study necessary (Zhu et al., 2025; Miller, 2024). Multiple works agree that a single success metric is insufficient, and that capabilities have to be judged both in the context of their task and in terms of their cost and reliability, broadening what is measured about an agent (Liang et al., 2023; Kapoor et al., 2026; 2025; Rabanser et al., 2026). We complement these works by treating the agent as a configurable system rather than a fixed model, and study how its configuration shapes these measurements.

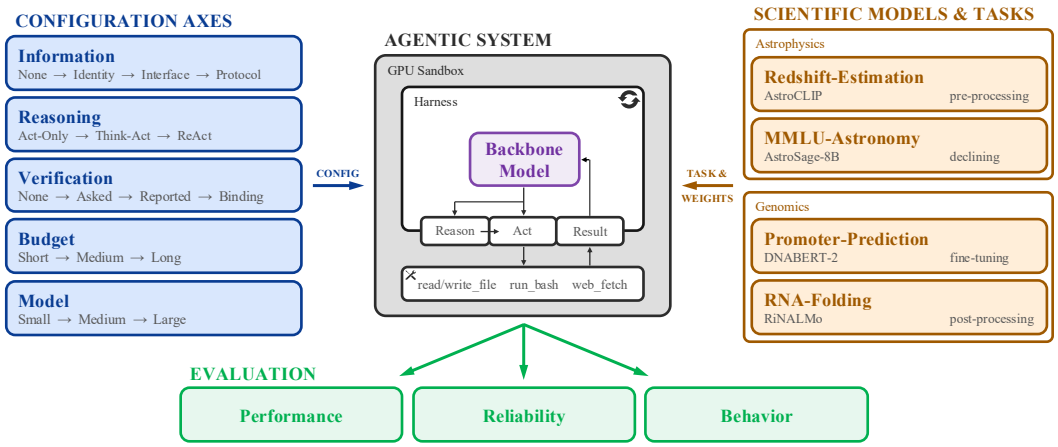


Figure 1: Architecture of our agentic AI4S benchmark. The agent (center) has access to different specialist models and is evaluated against their published performance (right). By varying the axes of the agent's configuration within the same task (left), we can systematically study its performance, reliability, and behavior under controlled, repeatable conditions (bottom).

3 Agent Evaluation and Benchmark Tasks

In our setting, a single agent has access to a collection of specialist models, including their weights and pre-built environments (i.e., the libraries and dependencies required to run them). Given a task, the agent has to determine which model to use (if any) and then correctly operate it by writing its own code. In the following, we describe how our agents and tasks are implemented (see Fig. 1 for an overview).

3.1 Modular Agent Implementation for Scientific Workflows

The agent runs in a sandbox with access to a single GPU and a pre-built coding environment. The sandbox contains the weights and environments of all specialist models, each with a short README that names the model and links to its paper and code. Importantly, the specialist models are not available through a dedicated tool call, but have to be operated by the agent the same way a human scientist would. The agent receives a simple system prompt that describes it as an "autonomous research agent" (Appendix A.16). It is then given the task description, the available data, and the required solution format. It can read and write files, fetch web pages, and run terminal commands, and can therefore write and execute its own code.

In each step, the agent's backbone model sees the conversation so far, including the task description, all outputs, and every tool result. Then, it either outputs plain text or calls one of the available tools (Appendix A.2). A tool's output is appended to the conversation before the loop repeats, so the agent always sees the outcome of its last action before choosing its next one. The loop ends when the agent calls a `finish` tool with its answer, or when the budget runs out. In its full form, this loop follows Yao et al. (2023), interleaving reasoning about what to do with the actions taken.

For our main analysis, we select and systematically vary five axes, each targeting a distinct factor known to shape behavior. What agents are told determines what they still need to explore or retrieve themselves (Jimenez et al., 2024). Keeping reasoning in context is central to the performance gains of Yao et al. (2023), and agents that check their own work tend to succeed more often (Shinn et al., 2023; Madaan et al., 2023). Giving agents more time lets them solve harder problems, though with diminishing returns (Wijk et al., 2025; Chan et al., 2025). Finally, the capability of the backbone model should be evaluated separately from the harness built around it (Kapoor et al., 2025). Together, the five axes cover what the agent knows, how it deliberates, whether it checks itself, how much time it has, and how capable it is, all of which can influence both outcome and reliability. We describe the axes and their levels below. For *Information* and *Verification*, each level adds to the previous one (marked with +).

Information. This axis controls how much information the agent receives in addition to the task description: (i) *none*: no additional information; (ii) *identity*: which specialist model is associated with the task; (iii) *interface*: + how to load and call the model; (iv) *protocol*: + a full recipe for this specific task.

Reasoning. This axis controls whether the agent generates reasoning and whether this reasoning stays in context for later steps: (i) *act-only*: no reasoning is generated; (ii) *think-act*: reasoning is generated in each step but discarded before the next; (iii) *ReAct*: reasoning is generated and kept in context across steps.

Verification. This axis controls how strongly the prompt asks the agent to verify its own answer: (i) *none*: no instruction to verify; (ii) *asked*: the agent is asked to verify its answer; (iii) *reported*: + the agent must state the score it expects and the evidence from its verification; (iv) *binding*: + the agent is told not to submit unless its own verification is convincing.

Budget. This axis controls how much wall-clock time the agent has to solve the task. We limit runtime rather than the number of steps, because the duration of a step varies widely (from a quick file read to fine-tuning a specialist model): (i) *short*: 5 min; (ii) *medium*: 10 min; (iii) *long*: 20 min.

Model. This axis controls the size, and therefore the capability, of the agent's backbone model: (i) *small*: Qwen3.5-35B-A3B; (ii) *medium*: Qwen3.5-122B-A10B; (iii) *large*: Qwen3.5-397B-A17B. All models are FP8-quantized due to resource limitations.

3.2 Benchmark Tasks from Astrophysics and Genomics

To create the tasks, we identify published scientific works that propose a new specialist model. For each, we reproduce the evaluation from the original publication ourselves, reducing its size where necessary so that it can be completed within the shortest budget (e.g., by downsampling the evaluation dataset). The score of the specialist model in our reproduction is the task's reference score. Each task tests, in a controlled setup, the agent's ability to understand, reproduce, and use prior work to complete a scientific workflow. We include the following tasks (Table 1 and Appendix A.17):

`redshift-estimation`. The agent has to estimate the cosmological redshift of galaxies directly from multi-band photometric images. Given 20 galaxy image cutouts and a labeled reference set, it has to use the pretrained AstroCLIP model (Parker et al., 2024) to predict continuous redshift values. Performance is measured by R^2 against spectroscopic ground-truth measurements.

`mmlu-astronomy`. This task consists of 152 multiple-choice questions on topics from stellar evolution to cosmology. The specialist model, AstroSage-8B (de Haan et al., 2025), is weaker than the agent's backbone model, so the task tests whether the agent recognizes this and answers the questions itself instead of relying on the specialist. Performance is measured by accuracy.

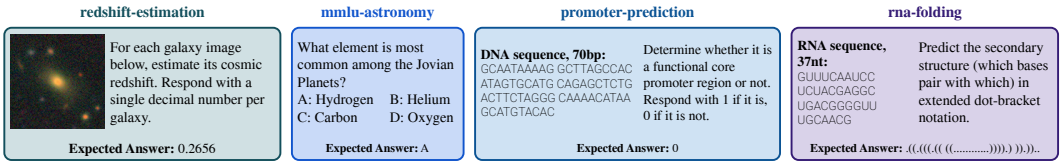
`promoter-prediction`. The agent has to classify 613 human DNA sequences as promoter or non-promoter regions. To do so, it has to locate, pre-process, fine-tune, and operate DNABERT-2 (Zhou et al., 2024). Performance is measured by the Matthews correlation coefficient (MCC).

`rna-folding`. The agent has to predict the secondary structures of 300 RNA sequences. It has to use the pretrained RiNALMo model (Penić et al., 2025) to predict contacts between nucleotide pairs, and then apply a post-processing procedure to obtain valid secondary structures. Performance is measured by the structure-level pairing F_1 score.

We distinguish two regimes: a task is GAP_POSITIVE if the specialist's reference score is higher than the score of the backbone model alone ($\mathcal{R} > \mathcal{B}$), and GAP_NEGATIVE if it is lower. The four tasks each stress a different part of using a specialist model correctly. `redshift-estimation` and `rna-folding` reward finding a stronger specialist and using it without fine-tuning, but the difficult step differs (pre-processing input images before embedding them, and post-processing the predicted contacts into valid structures). `promoter-prediction` instead requires fine-tuning DNABERT-2, because its pretrained checkpoint is not trained for promoter classification. `mmlu-astronomy` is GAP_NEGATIVE: its specialist model has been surpassed by general-purpose LLMs and is weaker than the backbone, so the correct behavior is to decline the specialist rather than use it. With two domains, two regimes, and four different demands on the specialist model, no single fixed strategy (always use the model, or never use it) performs well across all four tasks. To succeed, the agent has to investigate and judge each specialist's capabilities and work out its specific operating details.

Table 1: The four tasks of the benchmark, together with one example instance per task. Each task belongs to one of two domains and one of two regimes, and focuses on a different aspect of the specialist models' operation.

	`redshift-estimation`	`mmlu-astronomy`	`promoter-prediction`	`rna-folding`
Domain	Astrophysics	Astrophysics	Genomics	Genomics
Size	20 galaxy images	152 questions	613 DNA sequences	300 RNA sequences
Metric	R^2	Accuracy	MCC	F_1
Specialist	AstroCLIP	AstroSage-8B	DNABERT-2	RiNALMo
Challenge	Pre-processing	Declining	Fine-tuning	Post-processing
Regime	GAP_POSITIVE	GAP_NEGATIVE	GAP_POSITIVE	GAP_POSITIVE



4 Experiments and Results

4.1 Experimental Setup and Evaluation Metrics

To evaluate the agent, we use three quantities. $\mathcal{B}$ is the score of the backbone model alone, i.e., the LLM answering the task directly, without an agent or specialist model. $\mathcal{R}$ is the reference score of the specialist model in our reproduction. $\mathcal{S}$ is the agent's score on the task. Our main metric is the share of the gap between backbone and specialist that the agent closes (GAP_CLOSED):

$$\mathcal{G} = \frac{\mathcal{S} - \mathcal{B}}{\mathcal{R} - \mathcal{B}}, \tag{1}$$

where $\mathcal{G} = 0$ means no improvement over the backbone alone and $\mathcal{G} = 1$ means the agent matches the specialist's reference score. It is defined only for GAP_POSITIVE tasks ($\mathcal{R} > \mathcal{B}$). A run *completes* if the agent submits an answer with the `finish` tool before its budget runs out; runs that do not complete receive no score. For every completed run, we also record a binary *hurdle* indicator $\mathcal{H}$, which states whether the run's score exceeds the trivial score $\mathcal{T}$ of the task by more than a margin μ (Appendix A.14). $\mathcal{H}$ thus captures whether a run makes a genuine attempt at the task, and $\mathcal{G}$ captures how close it comes to the reference. For each task, we evaluate the full grid over our axes: 4 (*Information*) $\times$ 3 (*Reasoning*) $\times$ 4 (*Verification*) $\times$ 3 (*Budget*) $\times$ 3 (*Model*) $=$ 432 configurations, which we call *cells*. To measure how much outcomes vary between runs, we repeat every cell 5 times, which gives 2,160 runs per task and $4 \times 2{,}160 = 8{,}640$ runs in total for our main experiments.

4.2 Results

The four tasks differ widely in both score and reliability (Table 2). `promoter-prediction` is the most reliable: almost all of its completed runs clear the hurdle, and repeated runs of the same configuration agree closely. `redshift-estimation` is by far the least reliable, with more than half of its completed runs failing to clear the hurdle. Unless noted otherwise, the remaining analyses pool the three GAP_POSITIVE tasks, for which $\mathcal{G}$ is defined.

Agent outcomes are noisy. Even among genuine attempts, about 54% of the score variance is run-to-run noise rather than an effect of the configuration (Fig. 2, right). Without excluding runs that fail the hurdle, the noise appears even larger: differences between configurations then explain only 39.4% of the variance, instead of 46.1%. This noise is not uniform: cells with higher scores have systematically lower variance (Fig. 2, left). Whether a run clears the hurdle at all is also inconsistent across repeated runs, and this differs strongly between tasks: the pooled outcome consistency is 0.86, but only 0.73 for `redshift-estimation`, whose repeated runs disagree far more often than those of the other tasks (Tables 2 and 8). Even with 5 runs per cell, the 95% confidence interval of a cell's mean score remains wide (Appendix A.3). Given the level of run-to-run noise and the uneven

Table 2: Results per task for the main experiments (432 cells × 5 runs). Completion rate is computed over all runs; all other outcome rows are computed over completed runs. C_{out} is computed over cells with at least 3 completed runs. Full breakdown in Table 8. †Raw scale, not comparable across tasks.

	redshift-estimation	mmlu-astronomy	promoter-prediction	rna-folding
Runs & Completion				
# Runs	2,160	2,160	2,160	2,160
Completion Rate	76.3% ± 1.8%	95.3% ± 0.9%	89.3% ± 1.3%	61.7% ± 2.0%
% Cells that (always / mixed / never) complete	54.6 / 40.3 / 5.1	81.9 / 18.1 / 0.0	63.7 / 36.1 / 0.2	21.3 / 72.0 / 6.7
Anchors				
REFERENCE $\mathcal{R}$	0.753	0.671	0.746	0.735
BACKBONE $\mathcal{B}$	−2.701	0.967	−0.030	0.000
TRIVIAL $\mathcal{T}$	0.000	0.250	0.000	0.000
MARGIN μ	0.250	0.050	0.100	0.100
Outcome				
Mean Score $\mathcal{S}^{\dagger}$	−0.109	0.685	0.625	0.628
Mean GAP_CLOSED $\mathcal{G}$	0.750	—	0.844	0.854
Hurdle Failure Rate	53.8% ± 2.4%	4.7% ± 0.9%	0.5% ± 0.4%	8.7% ± 1.5%
Outcome Consistency C_{out}	0.727	0.905	0.986	0.825

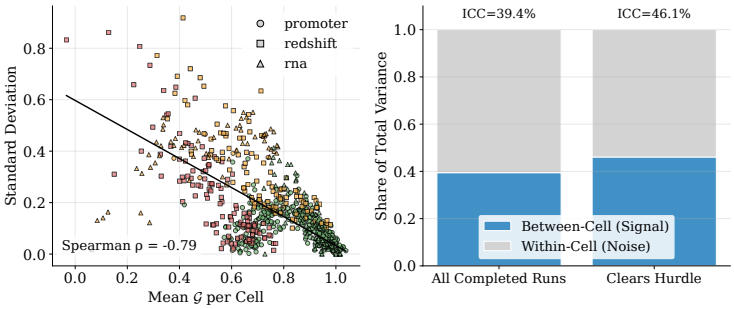


Figure 2: Outcome variance across cells. *Left*: mean $\mathcal{G}$ vs. SD per cell ($\geq$3 completed runs), with Spearman correlation. Color: all/some/no runs clear the hurdle (green/orange/red). *Right*: variance between vs. within cells, for all completed runs and hurdle-clearing runs only.

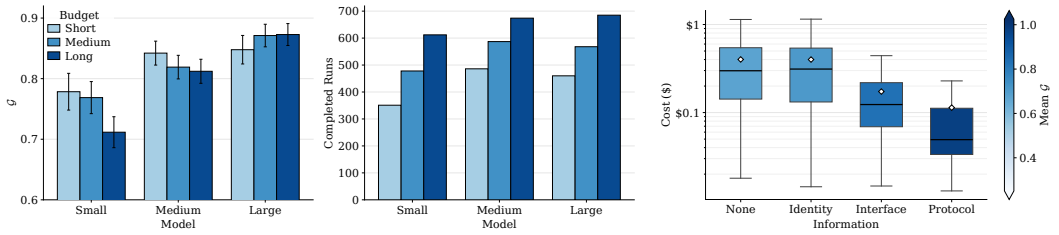
completion rates across configurations and tasks, our results reinforce recent calls for more rigorous reporting of completion rates, variance, and confidence intervals alongside raw scores as standard practice in agent evaluations (Kapoor et al., 2025; Rabanser et al., 2026).

Information has the largest axis effect. We rank the five axes by effect size: the range of an axis's mean $\mathcal{G}$ across its levels, divided by the within-cell standard deviation. *Information* has the largest effect on every task and clearly outranks *Model* (Table 3). On average, *Reasoning* matters more than *Budget*, and *Verification* ranks last. Because the levels of an axis complete at different rates, a ranking computed over completed runs can be affected by which runs complete. The ranking is nevertheless robust: *Information* ranks first in at least 99% of bootstrap samples, and a linear model gives the same order (Appendix A.9). This suggests that a user configuring an agent may benefit more from prioritizing task information than from choosing a stronger backbone model or a larger budget. The effect of one axis can, however, depend on the others, which we examine next.

The effect of one axis depends on the others. Increasing the *Budget* for the small and medium *Model* raises the completion rate but lowers $\mathcal{G}$, while for the large *Model* it raises both (Fig. 3a). *Information* shows a similar pattern: at lower levels, more *Budget* raises the completion rate without raising $\mathcal{G}$, while at the *interface* level it raises both (Fig. 6a). These patterns largely hold when we compare only configurations that complete under every budget, and the largest interaction overall is between *Information* and *Model* (Appendices A.9 and A.10). One explanation is that additional budget only improves the score once the agent can use it productively, either because it knows what to do (higher *Information*) or because it is capable enough (a larger *Model*). Without either, more time may mainly let the agent continue an unproductive approach until it submits, which would raise the completion rate without raising the score. The `redshift-estimation` case study matches this explanation: at the *none* level, more budget lowers the score (Table 9). One axis can therefore change not only how strongly another axis matters, but also the direction of its effect, and whether

Table 3: Effect of each axis on $\mathcal{G}$ per task (range of mean $\mathcal{G}$ across the axis's levels, divided by the within-cell standard deviation). Axes are ranked within each task (1 = largest effect).

	`promoter-prediction`		`redshift-estimation`		`rna-folding`		
Axis	Rank	Effect	Rank	Effect	Rank	Effect	Mean Rank
Information	1	1.925	1	1.452	1	1.496	1.00
Model	2	0.704	4	0.170	2	1.253	2.67
Reasoning	3	0.274	3	0.172	3	0.302	3.00
Budget	4	0.142	2	0.201	4	0.255	3.33
Verification	5	0.131	5	0.130	5	0.077	5.00



(a) Interaction of the *Model* and *Budget* axes.

(b) Axis effect of *Information* on task cost.

Figure 3: *Left*: $\mathcal{G}$ and number of completed runs per *Model* and *Budget*; error bars show 95% CI of the mean. *Right*: distribution of a run's cost per *Information* level, colored by that level's mean $\mathcal{G}$. Additional plots are in Fig. 6.

it moves completion and score in the same way. This supports evaluating agents as configurable systems rather than fixed models: how the parts combine matters as much as any single part.

Axes affect more than completion and score. Agents should be evaluated not only on their score but also on accompanying metrics such as cost and runtime (Kapoor et al., 2025). Lower *Information* levels not only reach a lower $\mathcal{G}$, but also cost more and run longer (Figs. 3b and 6b). We also measure calibration error: the score the agent predicts for itself minus its actual score, normalized by $|\mathcal{R} - \mathcal{B}|$, so that positive values mean the agent overestimates its result.[1] On average, agents are overconfident. *Reasoning* has virtually no effect on calibration error, and the small model is more overconfident than the medium and large models, which behave similarly. More *Information* reduces the calibration error, most strongly at the *protocol* level, where both its mean and its spread drop sharply (Fig. 7 and Appendix A.6). *Information* therefore improves $\mathcal{G}$, lowers cost and runtime, and improves calibration at the same time, with no trade-off, making it worth the effort to provide.

Agents rarely decline a weaker specialist. On the GAP_NEGATIVE task `mmlu-astronomy`, agents should answer with their backbone instead of the weaker specialist. Yet 99.8% of runs use AstroSage, including almost all runs at the *none* level, where it is not mentioned, and the mean accuracy stays close to the specialist's (Appendix A.13). Even with the oracle tool (Section 4.3), which lets the agent check the accuracy of its answers, performance does not increase substantially (Table 12). Possible explanations are that agents assume a specialist model outperforms a general one, similar to an authority bias (Xie et al., 2024), or that they explore too little, e.g., by never testing the backbone's own answers. Investigating these is an interesting direction for future work.

Summary. Agent outcomes on long, specialized tasks are noisy: about 54% of the score variance among genuine attempts is run-to-run noise rather than an effect of the configuration, and completion and outcome consistency vary sharply between tasks. A single run's score is therefore not a reliable signal on its own. Judging an axis only by its effect on $\mathcal{G}$ is also incomplete. *Information* has the largest effect on $\mathcal{G}$, and at the same time lowers cost and runtime and improves calibration. *Budget*, in contrast, mainly raises $\mathcal{G}$ when the agent has enough information or a large enough model to use the extra time productively. Agents also rarely question a weaker specialist. Building and evaluating

[1] Agents report a predicted score only at the *reported* and *binding* levels of *Verification*.

agents for such tasks should therefore track completion, score, cost, runtime, and calibration jointly across the axes that shape them, rather than optimizing a single axis for a single metric.

4.3 Ablations

Different model families behave similarly, but are not interchangeable. We rerun the experimental grid with an open-weight model from a different family, Step-3.7-Flash (comparable to the medium Qwen model, Qwen-122B), and with a closed model, Claude Sonnet 5 (roughly comparable to the large Qwen model, Qwen-397B). Both open-weight models rank *Information* first, while the order of the other axes shifts slightly (Table 14). Step-3.7-Flash also reaches a higher $\mathcal{G}$ than Qwen-122B at about half the cost per run (Table 15). The comparison between open-weight and closed model shows a sharper difference. The order of the other axes stays the same, but *Budget* rises from third to first rank for Sonnet 5 (Table 16). *Information* still matters, but less than time: the closed model is markedly more sensitive to how long it runs than to how much it is told. Sonnet 5 also executes far more efficiently: pooled across tasks, it needs less than half the tool calls of Qwen-397B, produces about a fifth of the tool-call errors, and is better calibrated. Yet at list prices, it costs more, and its pooled $\mathcal{G}$ is lower (Table 17). Efficient, low-error execution thus does not by itself lead to a better outcome, analogous to the decoupling of capability and reliability described by Rabanser et al. (2026). A system tuned to one model's characteristics therefore does not automatically transfer to another, necessitating a systematic analysis like ours to inform the system's priorities.

Verification works better at the system level. The *Verification* axis only prompts the agent to check its own work, as a user would, and its effect ranks last among all five axes (Table 3). To test whether verification works better when it is provided by the system instead, we equip the agent with an *oracle* tool that it can call at any point to score its current submission against the task's reference solution. The oracle represents the strongest form of system-level verification; in practice, a system would offer weaker checks, such as a validation set. We rerun the full grid with the oracle for two models, Qwen-122B and Step-3.7-Flash, with all axes varied as in the main experiments (Table 12). The oracle raises the score for both models on every task. It also sharply reduces the calibration error, which the prompted *Verification* axis does not. These gains are not free: runtime and cost rise in every case, because the agent makes more tool calls. Since the agent could also use the oracle to search for the correct answer, we checked all oracle runs for such hill-climbing. It occurs in only 4.4% of runs, and excluding them barely changes the results (Appendix A.12). This indicates that some agent behaviors, and the outcomes that depend on them, are easier to influence through the system than through the user's prompt.

5 Analysis: Taxonomy of Agent Behavior

Scores show whether an agent succeeds, but not how it behaves or why it fails. To study this, we automatically extract and classify the agents' outcomes and the behaviors that lead to them from the trajectories, using the following pipeline:

Stage 1: Structured Per-Trajectory Extraction. For every trajectory, we prompt a judge model, DeepSeek-V4-Flash-0731, to describe the trajectory along six dimensions: (i) an overall *result* characterization, (ii) the single root *error category* when applicable, (iii) specialist *model usage*, (iv) *verification behavior*, (v) *planning and exploration*, and (vi) *execution quality*. For every dimension, the judge receives the same context: the task description, the complete trajectory log, automatically derived facts about model usage and scoring, the task's intended solution approach, and the agent's final workspace. Following the extract-then-cluster method of Chirkova et al. (2026), each dimension is extracted as domain-agnostic free text rather than assigned from a predefined set of labels. The prompts are in Appendix A.18.

Stage 2: Clustering and Category Induction. We then consolidate the free-text descriptions of each dimension into a small set of general behavioral categories in three steps. First, we embed the descriptions with a Sentence-BERT model and cluster them with HDBSCAN; points that HDBSCAN leaves unassigned are added to their most similar cluster. Second, the judge model labels each cluster with a short, general description of the behavior, based on a representative sample of its members. Task-specific details go into a separate criterion rather than into the label. Third, the

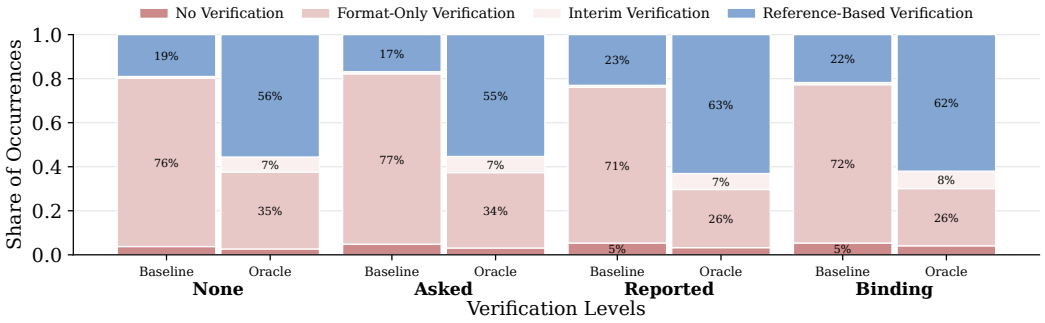


Figure 4: Distribution of verification behavior categories per level of the *Verification* axis, without (Baseline) and with the *oracle* tool, pooled across Qwen-122B, Step-3.7-Flash, and all tasks.

judge model reviews all cluster labels of one dimension at once and merges them into at most $K = 8$ general categories, combining near-duplicates and overly specific labels. In practice, no dimension reached this limit. As a result, every trajectory receives one category per dimension, which we cross-tabulate against the experimental conditions (details in Appendix A.15).

Results. Across all dimensions, *Information* has the strongest association with the distribution of categories (Table 24), consistent with its dominant effect on scores. We highlight one case study that connects to the previous section: *verification behavior*. The prompted *Verification* axis had the smallest effect on the score (Table 3), while the oracle tool raised the score for both models on every task (Section 4.3). The taxonomy shows how the agents' behavior differs between the two. Without the oracle, prompting barely changes how agents verify: from *none* to *binding*, the share of reference-based verification rises only from 19% to 22%, and about three quarters of runs check only the format of their answer (Fig. 4). With the oracle, the share of reference-based verification roughly triples at every prompt level. Part of this shift reflects the oracle calls themselves, but it shows that agents use a verification option when the system offers one, largely regardless of what the prompt asks for. This illustrates a broader point: some agent behaviors are hard to shift through the user's prompt but respond strongly to what the system provides, and the system should be designed with this distinction in mind.

6 Conclusion

We systematically study how five axes of an agent's configuration influence its outcomes on scientific tasks that require operating specialist models. Four findings stand out. First, agent outcomes are noisy: about 54% of the score variance among genuine attempts is run-to-run noise rather than an effect of the configuration. Second, in our main experiments, the information given to the agent has the largest effect on the score, outranking both the backbone model and the budget, and it also lowers cost and runtime and improves calibration. Third, axes interact: additional budget mainly raises the score when the agent has enough information or a large enough model to use it productively, and can lower the score otherwise, so axes cannot be tuned in isolation. Fourth, some agent behaviors, such as self-verification, barely respond to prompting by the user but shift strongly when the system provides a dedicated tool. Together, these results argue for evaluating and building agents as configurable systems rather than fixed models: tracking completion, variance, cost, and calibration alongside raw scores, and designing the system around behaviors that prompting alone cannot reliably control.

Limitations. Our analysis covers over 18,000 agent trajectories, but is limited to four tasks across two scientific domains. More tasks and domains would strengthen the generality of the conclusions. We exclude several further system components (memory, multi-agent coordination) that future work should vary as additional axes. Finally, our finding that some agent behaviors respond more to system-level design than to user prompting is demonstrated on one dimension (verification) and needs to be tested across other behavioral dimensions before it can be treated as a general principle.

AI USE STATEMENT

In this work, we used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, and interpret results. We have not used generative AI tools (or it is not applicable to this work) to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims or assist in the writing of proofs. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, draft parts of a research paper, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, identify relevant literature, and propose a title or keywords for a research paper. We have reviewed all AI-assisted work. All AI use was exclusively used as a first step, and went through manual checks for accuracy, correctness, potential plagiarism, and understandability. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To make the paper fully reproducible, we will release all code and instructions on how to use it upon acceptance. We will additionally release all produced artifacts (agent traces) and derived analyses (raw taxonomy data) to encourage further usage in the community.

REFERENCES

Alejandro H Artiles, Martin Weiss, Levin Brinkmann, Iyad Rahwan, Bernhard Schölkopf, Christopher Pal, Hugo Larochelle, Anirudh Goyal, and Nasim Rahaman. The alien space of science: Sampling coherent but cognitively unavailable research directions. In *arXiv*, 2026.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. Super: Evaluating agents on setting up and executing tasks from research repositories. In *EMNLP*, 2024.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *NeurIPS*, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. In *ACM transactions on intelligent systems and technology*, 2024.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *ICLR*, 2025.

Nadezhda Chirkova, Tunde Oluwaseyi Ajayi, Seth Aycock, Zain Muhammad Mujahid, Vladana Perlić, Ekaterina Borisova, and Markarit Vartampetian. Llm-as-a-qualitative-judge: automating error analysis in natural language generation. In *First Workshop on Multilingual Multicultural Evaluation*, 2026.

John G. Cragg. Some statistical models for limited dependent variables with application to the demand for durable goods. In *Econometrica*, 1971.

Tijmen de Haan, Yuan-Sen Ting, Tirthankar Ghosal, Tuan Dung Nguyen, Alberto Accomazzi, Azton Wells, Nesar Ramachandra, Rui Pan, and Zechang Sun. Achieving gpt-4o level performance in astronomy with a specialized 8b-parameter large language model. In *Scientific Reports*, 2025.

Sanderson Oliveira de Macedo. What makes a harness a harness: necessary and sufficient conditions for an agent harness. In *arXiv*, 2026.

Damon Falck, Samer Sabri, Anja Surina, Thom Foster, Anya Sims, Sam Devlin, Dylan Rogers, Tantum Collins, Kaloyan Aleksiev, Louis Kirsch, et al. Training ai scientists to replicate research. In *arXiv*, 2026.

Mourad Gridach, Jay Nanavati, Christina Mack, Khaldoun Zine El Abidine, and Lenon Mendes. Agentic AI for scientific discovery: A survey of progress, challenges, and future directions. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. In *Nature*, 2021.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. In *TMLR*, 2025.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *ICLR*, 2026.

Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. Beyond pass@ 1: A reliability science framework for long-horizon llm agents. In *arXiv*, 2026.

Gyeongwon J Kim, Alex Wilf, Louis-Philippe Morency, and Daniel Fried. From reproduction to replication: Evaluating research agents with progressive code masking. In *ICLR*, 2026.

Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, et al. Can ai agents conduct open-ended ai research? early evidence from two case studies. In *arXiv*, 2026.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long software tasks. In *NeurIPS*, 2025.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. In *TMLR*, 2023.

Ryan Lopopolo. Harness engineering: Leveraging Codex in an agent-first world. OpenAI, 2026. URL `https://openai.com/index/harness-engineering/`. Published 2026-02-11.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. In *Nature*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

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. In *ICLR*, 2025.

Lisa Messeri and Molly J Crockett. Artificial intelligence and illusions of understanding in scientific research. In *Nature*, 2024.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. In *arXiv*, 2024.

Sergio Paniego and Aritra Roy Gosthipaty. Harness, scaffold, and the AI agent terms worth getting right. Hugging Face Blog, 2026. URL `https://huggingface.co/blog/agent-glossary`. Published 2026-05-25.

Liam Parker, Francois Lanusse, Siavash Golkar, Leopoldo Sarra, Miles Cranmer, Alberto Bietti, Michael Eickenberg, Geraud Krawezik, Michael McCabe, Rudy Morel, et al. Astroclip: a cross-modal foundation model for galaxies. In *Monthly Notices of the Royal Astronomical Society*, 2024.

Rafael Josip Penić, Tin Vlašić, Roland G Huber, Yue Wan, and Mile Šikić. Rinalmo: general-purpose rna language models can generalize well on structure prediction tasks. In *Nature Communications*, 2025.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. In *ICML*, 2026.

Yujiong Shen, Yajie Yang, Zhiheng Xi, Binze Hu, Huayu Sha, Jiazheng Zhang, Qiyuan Peng, Junlin Shang, Jixuan Huang, Yutao Fan, et al. Sciagentgym: Benchmarking multi-step scientific tool-use in llm agents. In *arXiv*, 2026.

Yifan Shi, Wei Zhang, and Tianyi Cui. A programming paradigm for spatiotemporal composability. In *arXiv*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? a large-scale human study with 100+ NLP researchers. In *ICLR*, 2025.

Chenglei Si, Tatsunori Hashimoto, and Diyi Yang. The ideation-execution gap: Execution outcomes of LLM-generated versus human research ideas. In *ICLR*, 2026.

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. In *ICML*, 2025.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. In *Nature*, 2023.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, et al. Naturebench: Can coding agents match the published sota of nature-family papers? In *arXiv*, 2026.

Jiaqi Wei, Yuejin Yang, Xiang Zhang, Yuhan Chen, Xiang Zhuang, Zhangyang Gao, Dongzhan Zhou, Guangshuai Wang, Zhiqiang Gao, Juntai Cao, et al. From ai for science to agentic science: A survey on autonomous scientific discovery. In *arXiv*, 2025.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. In *ICML*, 2025.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. In *Journal of the American Statistical Association*, 1927.

Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. In *The knowledge engineering review*, 1995.

Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *ICLR*, 2024.

Hongliang Xin, John R Kitchin, and Heather J Kulik. Towards agentic science for advancing scientific discovery. In *Nature Machine Intelligence*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Published 2025-11-26.

Tom Zahavy. Position: LLMs can't jump. In *ICML*, 2026.

Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. Dnabert-2: Efficient foundation model and benchmark for multi-species genomes. In *ICLR*, 2024.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, et al. Establishing best practices in building rigorous agentic benchmarks. In *NeurIPS*, volume 38, 2025.

A Appendix

A.1 Implementation Details

We serve each open-weight model with vLLM, using tensor parallelism across its GPUs: one NVIDIA H100 GPU for the small Qwen model, two for the medium, eight for the large, and four for Step-3.7-Flash. All models are FP8-quantized and use a context window of 262,144 (Qwen) or 256,000 (Step) tokens. Claude Sonnet 5 is accessed through the Anthropic API. Each cell runs as a separate SLURM array job against the shared model server, with up to 8 concurrent agent runs, each in its own sandbox with access to a single GPU. Table 4 lists the composition of the released trajectories.

Table 4: Composition of the released trajectories. Each cell is repeated 5 times and the grid covers all four tasks.

Grid	Cells per task	Runs per task	Trajectories
Main experiments (3 Qwen3.5 models)	432	2,160	8,640
Step-3.7-Flash (no *act-only*)	96	480	1,920
Claude Sonnet 5	144	720	2,880
Oracle, Qwen-122B	144	720	2,880
Oracle, Step-3.7-Flash	96	480	1,920
Total			18,240

A.2 Tool Use and Specialist Model Usage

Table 5 shows, for each tool, the mean number of calls per run and the share of these calls that returned an error, for the main experiments. A `run_bash` call counts as an error if the command does not exit with code 0. A call to any other tool counts as an error if the harness reports that it could not execute it (e.g., a missing file or an unreachable URL). *Other/malformed* covers all calls to tool names the harness did not recognize, which always fail. `run_bash` accounts for most calls on every task, while the use of `web_fetch` differs substantially between tasks.

Table 5: Mean tool calls per run and error rate per tool, main experiments.

	`run_bash`		`read_file`		`write_file`		`web_fetch`		`finish`		Other/malformed		Total	
Task	Calls	Err.%	Calls	Err.%	Calls	Err.%	Calls	Err.%	Calls	Err.%	Calls	Err.%	Calls	Err.%
`redshift-estimation`	28.51	11.4	4.24	5.7	3.56	3.9	0.79	0.0	0.76	0.1	0.07	100.0	37.93	9.7
`mmlu-astronomy`	17.84	12.7	2.92	1.3	2.36	1.4	0.07	3.8	0.95	0.0	0.05	100.0	24.20	9.9
`promoter-prediction`	16.31	16.2	3.09	0.7	3.73	3.7	0.04	4.3	0.89	0.1	0.04	100.0	24.10	11.8
`rna-folding`	33.21	14.2	3.73	4.0	6.14	3.4	1.62	0.7	0.62	0.3	0.18	100.0	45.49	11.6

Table 6 breaks the `web_fetch` calls down by destination. *Ref. correct* and *Ref. decoy* give the share of fetches to a URL listed in the README of the correct specialist model or of a decoy. Decoys are the other specialist models in the sandbox, which do not fit the task; on `mmlu-astronomy`, we also count AstroSage as a decoy, because the correct behavior is not to use it. *Off-list* covers all other URLs. On the three GAP_POSITIVE tasks, 85–96% of fetches go to the correct model's reference material. The off-list remainder is mostly general troubleshooting (a search engine, Colab, PyPI, the GitHub API), except on `promoter-prediction`, where some off-list fetches to `huggingface.co` read similarly named but wrong repositories (e.g., `nlpaueb/dnabert-2-base-100M`) instead of the one provided. On `mmlu-astronomy`, most fetches (73%) also go to AstroSage's reference material, while 27% go elsewhere, mainly to a search engine and Wikipedia.

Table 6: `web_fetch` destinations, main experiments. HTTP OK % is the share of fetches that return a 2xx status. A fetch that returns 404 still counts as successful in Table 5, because the harness executed it without an error.

Task	Fetches/run	Ref. correct %	Ref. decoy %	Off-list %	HTTP OK %	Top off-list hosts
`redshift-estimation`	0.79	88.9	0.1	11.0	84.4	`colab.research.google.com`, `lite.duckduckgo.com`
`mmlu-astronomy`	0.07	0.0	73.0	27.0	87.4	`lite.duckduckgo.com`, `en.wikipedia.org`
`promoter-prediction`	0.04	84.8	0.0	15.2	85.9	`huggingface.co`, `raw.githubusercontent.com`
`rna-folding`	1.62	95.9	0.0	4.1	86.7	`api.github.com`, `lite.duckduckgo.com`

Table 7 shows how agents engage with each specialist model in the main experiments, based on patterns in the trajectories. It gives the share of runs in which a model's directory is *mentioned* in a tool call, its documentation is read (*read_docs*), a command that names the model's path runs and exits with code 0 (*executed*), and the agent fetches a URL from the model's README (*researched_online*). On the three GAP_POSITIVE tasks, the correct specialist model is read and executed in 95–99% of runs, while no decoy is executed in more than 1% of runs. `mmlu-astronomy` has no correct specialist model, since it can be answered directly by the backbone. Still, the weaker AstroSage is read in 98.6% of runs and executed in 66.1%. Because *executed* only counts commands that name the model's path, it undercounts use: counting scripts that load AstroSage as well, 99.8% of runs use it (Appendix A.13).

Table 7: Engagement with each specialist model (% of runs), main experiments. The correct specialist model per task is in bold, `mmlu-astronomy` has none. `redshift-estimation` and `mmlu-astronomy` were run before the genomics models were available to the agent, however, the genomics tasks, whose sandbox contained all four models, show that the additional models do not keep the agent from finding the right one.

Task	Specialist	*mentioned*	*read_docs*	*executed*	*researched_online*
`redshift-estimation`	**`astroclip`**	100.0%	99.9%	99.3%	28.3%
	`astrosage`	27.3%	27.1%	0.2%	0.1%
	`dnabert-2`	0.0%	0.0%	0.0%	0.0%
	`rinalmo`	0.0%	0.0%	0.0%	0.0%
`mmlu-astronomy`	`astroclip`	30.5%	30.2%	0.3%	0.0%
	`astrosage`	100.0%	98.6%	66.1%	5.0%
	`dnabert-2`	0.0%	0.0%	0.0%	0.0%
	`rinalmo`	0.0%	0.0%	0.0%	0.0%
`promoter-prediction`	**`dnabert-2`**	100.0%	99.8%	94.7%	3.6%
	`astroclip`	11.9%	11.9%	0.0%	0.0%
	`astrosage`	10.1%	10.1%	0.0%	0.0%
	`rinalmo`	24.3%	24.3%	0.2%	0.0%
`rna-folding`	**`rinalmo`**	100.0%	100.0%	99.3%	38.2%
	`astroclip`	10.3%	9.3%	0.1%	0.0%
	`astrosage`	8.9%	8.3%	0.0%	0.0%
	`dnabert-2`	14.6%	14.0%	0.1%	0.0%

A.3 VARIANCE DECOMPOSITION AND HURDLE

Table 8: Hurdle decomposition and run-to-run variance per task. Mean per-cell SD and the 95% CI half-width of a cell's mean score (at 5 runs) are expressed as % of $|\mathcal{R} - \mathcal{T}|$, so they are comparable across tasks. The pooled CI half-width in units of $\mathcal{G}$ is shown in Fig. 5 (right).

Task	Hurdle failure rate (95% CI)	C_{out}	ICC, all runs	ICC, clears hurdle	Mean per-cell SD	95% CI (5 runs)
`redshift-estimation`	53.8% [51.4, 56.2]	0.727	33.3%	36.2%	85.1%	91.9%
`mmlu-astronomy`	4.7% [3.9, 5.7]	0.905	29.4%	6.9%	17.6%	20.8%
`promoter-prediction`	0.5% [0.2, 0.9]	0.986	47.9%	49.7%	10.7%	12.9%
`rna-folding`	8.7% [7.3, 10.3]	0.825	45.7%	35.2%	12.8%	13.0%
Pooled (3 GAP_POSITIVE tasks, task means removed)	20.6% [19.5, 21.7]	0.858	39.4%	46.1%	—	—

Intraclass correlation. To separate real signal between configurations from run-to-run noise, we decompose the score variance in two steps. First, a one-way random-effects ANOVA over all cells with at least three completed runs splits the total variance into a within-cell component (MS_{within}) and a bias-corrected between-cell component. From these, we compute the intraclass correlation

$$\text{ICC} = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{between}} + \sigma^2_{\text{within}}},$$

the share of the total variance that comes from differences between cells rather than from run-to-run noise. On its own, this decomposition is misleading: a run can complete and still fail to make a genuine attempt at the task (e.g., by running the wrong specialist model or submitting predictions far below the trivial score). Because the ANOVA sums squared deviations, such rare but severe failures inflate MS_{within} far out of proportion to how often they occur. We therefore also compute

a two-part (hurdle) decomposition (Cragg, 1971), following recent work on the reliability of LLM agents across repeated trials that separates a binary success indicator per run from a continuous score given success (Rabanser et al., 2026; Khanal et al., 2026).

Hurdle indicator and failure rate. We write $y \in \{0, 1\}$ for the hurdle indicator $\mathcal{H}$ of a run: $y = 1$ if the run's score exceeds the trivial score $\mathcal{T}$ by more than the margin μ, and $y = 0$ otherwise. The hurdle is distinct from completion: runs that do not complete (e.g., because they run out of budget) have no score and are not counted here at all. The *hurdle failure rate* is the share of completed runs with $y = 0$, i.e., $1 - \bar{y}$. We report 95% Wilson score intervals (Wilson, 1927). The completion rate in Table 2 uses the same interval, computed over all 2,160 runs of a task. The $\pm$ values in Table 2 give the half-widths of these intervals, which are nearly symmetric at these sample sizes.

Outcome consistency. For each cell i with at least 3 completed runs, let $\hat{p}_i$ be the share of its runs that clear the hurdle. Following Rabanser et al. (2026),

$$C_{\text{out}} = \frac{1}{k}\sum_{i=1}^{k}(2\hat{p}_i - 1)^2 = \frac{1}{k}\sum_{i=1}^{k}\left[1 - \frac{\hat{p}_i(1-\hat{p}_i)}{0.25}\right],$$

averaged over the k qualifying cells. Each term normalizes the variance of the cell's binary outcomes by the maximum variance of a Bernoulli variable, 0.25. A cell contributes 1 if all its runs clear the hurdle or all fail it, and 0 if exactly half of them clear it. $C_{\text{out}} = 1$ therefore means that every configuration reliably succeeds or reliably fails.

Variance among genuine attempts. We apply the same ANOVA and ICC to runs with $y = 1$ only. This measures how repeatable an agent's score is, given that it makes a genuine attempt. Since this restriction selects runs by their outcome, it can bias the variance estimate, so we report both versions.

Pooling across tasks. Cells are defined per task. Pooling the raw $\mathcal{G}$ values of several tasks would therefore count differences between the task means as differences between cells, and inflate the pooled ICC above every per-task value. We therefore subtract each task's mean before pooling.

A.4 Additional Completion Analyses

Not every run produces a score, for example when the agent runs out of budget before submitting a solution. Across all four tasks, 80.7% of runs complete, and 19.3% run out of budget. Completion also varies within configurations: on every task, a substantial share of cells completes on some runs but not on others, from run-to-run variation alone (Table 2; Fig. 5, left, for the three GAP_POSITIVE tasks). Per-cell completion rates range from 0% to 100% on every task except `mmlu-astronomy`, where every configuration completes at least 20% of its runs. We flag a run as an extreme outlier if its score differs from the trivial score by more than 10 times the gap between reference and trivial score, $10\,|\mathcal{R} - \mathcal{T}|$. Such outliers occur only on `redshift-estimation` (7 of 1,647 completed runs, 0.4%), all but one with the small model. We exclude them from all score-based analyses.

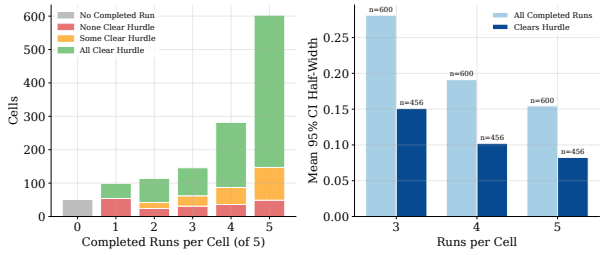
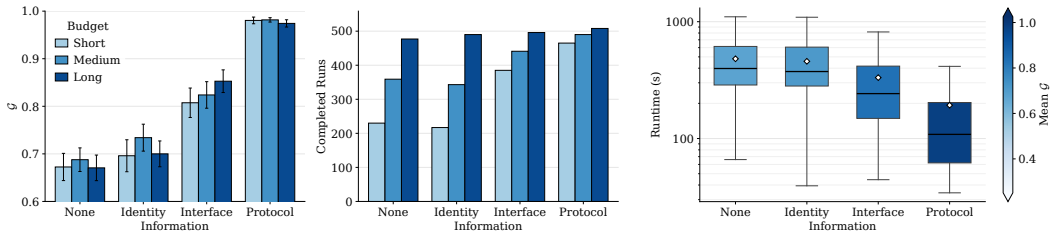


Figure 5: Completion and variance details, pooled across the three GAP_POSITIVE tasks. *Left*: cells by their number of completed runs (0–5); color shows whether all, some, or none of a cell's completed runs clear the hurdle (green/orange/red), and gray marks cells without any completed run. *Right*: mean 95% CI half-width of a cell's mean $\mathcal{G}$ at 3–5 runs, for all completed runs (light blue) and for hurdle-clearing runs only (dark blue).

A.5 Additional Interaction and Runtime Views

Figure 6 shows the *Information*×*Budget* interaction, next to the *Model*×*Budget* interaction in the main text (Fig. 3), and the effect of *Information* on runtime, next to its effect on cost.



(a) Interaction of the *Information* and *Budget* axes.

(b) Axis effect of *Information* on runtime.

Figure 6: *Left*: $\mathcal{G}$ and number of completed runs per *Information* and *Budget* level; error bars show 95% CI of the mean. *Right*: distribution of a run's wall-clock time per *Information* level, colored by that level's mean $\mathcal{G}$.

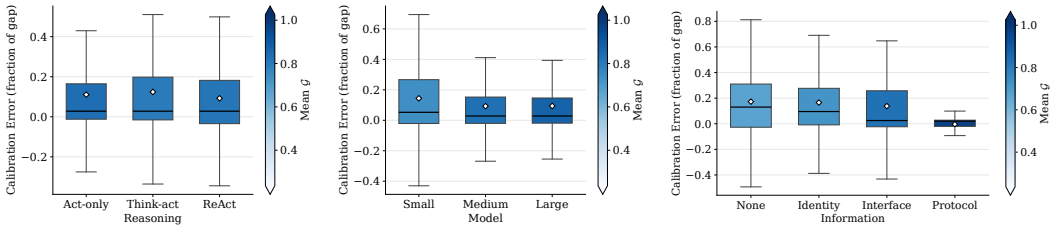
Case study: `redshift-estimation`. How axes interact depends on the task, but `redshift-estimation` offers a clear example (Table 9). At the *none* level, more *Budget* lowers the score ($-0.429 \rightarrow -0.713$) at similar hurdle failure rates, consistent with the agent pursuing a wrong approach further. At the *interface* level, more *Budget* raises the score ($-0.605 \rightarrow -0.380$) and lowers the hurdle failure rate: the agent now knows enough to use the extra time. At the *protocol* level, *Budget* makes no difference, because every run already reproduces the reference score. The medians show the same pattern (*none*: $-0.34 \rightarrow -0.50$; *interface*: $-0.55 \rightarrow -0.34$), and so does the comparison restricted to configurations that have completed runs under all three budgets (*none*: $-0.44 \rightarrow -0.77$; *interface*: $-0.61 \rightarrow -0.38$).

Table 9: Mean R^2 & $\mathcal{H}$ failure rate, Budget × Information, `redshift-estimation`.

Info/Budget	Short	Medium	Long
None	-0.429	-0.441	-0.713
	89.1%	92.0%	92.2%
Identity	-0.514	-0.519	-0.516
	83.8%	84.4%	84.4%
Interface	-0.605	-0.560	-0.380
	76.1%	71.9%	65.9%
Protocol	0.753	0.753	0.753
	0.0%	0.0%	0.0%

A.6 Calibration Error

Fig. 7 shows the calibration error, defined in Section 4 as the agent's predicted score minus its actual score, normalized by $|\mathcal{R} - \mathcal{B}|$, per level of *Reasoning*, *Model*, and *Information*.



(a) *Reasoning*

(b) *Model*

(c) *Information*

Figure 7: Calibration error per level of *Reasoning*, *Model*, and *Information*, pooled across the three GAP_POSITIVE tasks. Positive values mean the agent overestimates its score. Each box is colored by that level's mean $\mathcal{G}$.

A.7 Agent Effort and Cost

Table 10 gives the mean runtime, cost, steps, tool calls, and tool-call errors per run in the main experiments. Since we run the open-weight models on our own GPUs, we estimate their cost from

Table 10: Mean cost and effort per run in the main experiments (432 cells × 5 runs).

	redshift-estimation	mmlu-astronomy	promoter-prediction	rna-folding
Mean Runtime (s)	344.2	188.3	278.0	476.0
Mean Cost (USD)	0.277	0.107	0.115	0.424
Mean Steps / Tool Calls / Errors	33 / 38 / 4	21 / 24 / 2	20 / 24 / 3	41 / 45 / 5

Table 11: Cost of one full 720-run task grid, per model. Tokens shown as prompt / completion, in millions.

	Qwen-35B-A3B		Qwen-122B-A10B		Qwen-397B-A17B		Sonnet 5			
Task	Tokens	Cost	Tokens	Cost	Tokens	Cost	Tokens	Calculated	Real	Savings
redshift	659.5 / 10.3	$186	397.2 / 7.7	$183	340.1 / 7.0	$229	147.8 / 4.1	$337	$124	63.3%
mmlu	222.3 / 3.7	$63	142.0 / 3.2	$67	150.7 / 3.1	$102	65.4 / 2.8	$159	$66	58.6%
promoter	306.0 / 7.1	$91	143.0 / 4.9	$73	117.4 / 4.1	$85	88.5 / 3.9	$216	$84	60.9%
rna	978.5 / 15.3	$275	579.3 / 13.0	$273	530.3 / 13.5	$367	140.8 / 4.6	$328	$110	66.4%

token counts at the first-party API list prices (USD per 1M input/output tokens): $0.25/$2.00 for Qwen-35B, $0.40/$3.20 for Qwen-122B, and $0.60/$3.60 for Qwen-397B (Alibaba Cloud Model Studio), and $0.20/$1.15 for Step-3.7-Flash (StepFun). For Claude Sonnet 5, we use Anthropic's list price of $2/$10. Table 11 gives the total cost of one full grid per task and model. For Sonnet 5, *Calculated* prices its tokens at the list price without caching discounts, using the same formula as for the open-weight models, and is therefore the directly comparable number. *Real* is what Anthropic actually billed, with prompt caching, and *Savings* gives how much lower this is than *Calculated*.

A.8 Ablation Details

The outcome and cost tables in this subsection (Tables 12, 15 and 17) share the same conventions. mmlu-astronomy is GAP_NEGATIVE, so its row reports raw accuracy† instead of $\mathcal{G}$. Calibration error is normalized by $|\mathcal{R} - \mathcal{B}|$. Cost, wall-clock time, steps, tool calls, and errors are means per run, with costs estimated at list prices (Appendix A.7).

Oracle ablation. This paragraph gives the full results of the oracle ablation (Section 4.3). The agent receives an oracle_check tool, independent of the prompted *Verification* level, which scores its current submission against the task's reference solution on demand. We rerun the full grid with this tool for two models, Qwen-122B and Step-3.7-Flash, with all axes varied as in the main experiments. Table 12 compares outcome, calibration, cost, and effort with and without the tool, and Table 13 compares the axis ranks. The baseline for Qwen-122B uses its full grid, including *act-only*, while Table 15 excludes this level; the numbers for Qwen-122B in the two tables therefore differ slightly.

Model-family and open- vs. closed-weight ablations. This paragraph gives the full results of the two model ablations in Section 4.3. Tables 14 and 15 compare the two open-weight models, and Tables 16 and 17 compare an open-weight and a closed model. For each model, we repeat the axis-effect ranking (*Model* is fixed within each block and therefore omitted) and report outcome, calibration, cost, and effort.

Table 12: Oracle ablation: outcome, calibration error, cost/effort with vs. without the oracle_check tool.

		Outcome		Cal. Err.		Cost (USD)		Wall-clock (s)		Steps / TC / Err.	
Model	Task(s)	Baseline	Oracle	Baseline	Oracle	Baseline	Oracle	Baseline	Oracle	Baseline	Oracle
Qwen-122B	mmlu-astronomy†	0.707	0.725	0.152	−0.002	0.093	0.125	165	220	19 / 22 / 2	21 / 24 / 2
	Pooled GAP_POSITIVE	0.822	0.923	0.093	0.002	0.245	0.358	346	453	28 / 32 / 4	34 / 39 / 5
Step-3.7-Flash	mmlu-astronomy†	0.677	0.721	0.285	0.065	0.043	0.091	164	309	18 / 20 / 2	26 / 28 / 3
	Pooled GAP_POSITIVE	0.865	0.889	0.073	0.026	0.126	0.177	369	455	29 / 31 / 4	35 / 37 / 5

Table 13: Axis-effect ranks on $\mathcal{G}$ per task, without (Baseline) and with the oracle tool, for Qwen-122B-A10B and Step-3.7-Flash (1 = largest effect). Unlike Table 14, the Qwen-122B baseline includes the *act-only* level, so its ranks differ from that table.

		promoter		redshift		rna		Mean Rank	
Model	Axis	Baseline	Oracle	Baseline	Oracle	Baseline	Oracle	Baseline	Oracle
Qwen-122B	Information	1	1	1	1	1	1	1.00	1.00
	Reasoning	2	2	3	2	2	2	2.33	2.00
	Verification	3	4	4	4	3	3	3.33	3.67
	Budget	4	3	2	3	4	4	3.33	3.33
Step-3.7-Flash	Information	1	2	1	1	1	1	1.00	1.33
	Reasoning	3	3	2	3	2	4	2.33	3.33
	Verification	4	4	3	2	3	3	3.33	3.00
	Budget	2	1	4	4	4	2	3.33	2.33

Table 14: Axis effects on $\mathcal{G}$ for two open-weight models of similar size: Qwen-122B-A10B and Step-3.7-Flash (198B-A11B). Step-3.7-Flash does not support the *act-only* level of *Reasoning*, so both blocks exclude this level; *Reasoning*'s rank and effect are computed over its remaining two levels.

		promoter		redshift		rna		
Model	Axis	Rank	Effect	Rank	Effect	Rank	Effect	Mean Rank
Qwen-122B	Information	1	2.172	1	2.027	1	1.410	1.00
	Verification	2	0.256	3	0.195	3	0.193	2.67
	Budget	4	0.086	2	0.313	2	0.262	2.67
	Reasoning	3	0.116	4	0.028	4	0.041	3.67
Step-3.7-Flash	Information	1	0.848	1	0.951	1	1.412	1.00
	Reasoning	3	0.204	2	0.298	2	0.425	2.33
	Verification	4	0.173	3	0.282	3	0.247	3.33
	Budget	2	0.558	4	0.177	4	0.058	3.33

Table 15: Qwen-122B-A10B vs. Step-3.7-Flash: outcome, calibration error, cost, and effort, excluding *act-only* as in Table 14.

Model	Task(s)	Outcome	Cal. Err.	Cost (USD)	Wall-clock (s)	Steps / TC / Err.
Qwen-122B	mmlu-astronomy†	0.702	0.153	0.089	164	17.8 / 21.6 / 2.1
	Pooled GAP_POSITIVE	0.802	0.105	0.228	347	25.7 / 31.9 / 3.7
Step-3.7-Flash	mmlu-astronomy†	0.677	0.285	0.043	164	17.8 / 19.9 / 2.1
	Pooled GAP_POSITIVE	0.865	0.073	0.126	369	28.9 / 30.8 / 4.1

Table 16: Axis effects on $\mathcal{G}$ for an open-weight and a closed model: Qwen-397B-A17B vs. Claude Sonnet 5.

		promoter		redshift		rna		
Model	Axis	Rank	Effect	Rank	Effect	Rank	Effect	Mean Rank
Qwen-397B	Information	1	1.884	1	1.126	1	1.051	1.00
	Reasoning	3	0.467	2	0.238	2	0.233	2.33
	Budget	2	0.497	4	0.106	4	0.070	3.33
	Verification	4	0.235	3	0.207	3	0.178	3.33
Claude Sonnet 5	Budget	1	1.589	1	1.878	1	1.796	1.00
	Information	2	0.521	2	1.293	2	1.683	2.00
	Reasoning	3	0.287	3	0.370	4	0.219	3.33
	Verification	4	0.099	4	0.153	3	0.279	3.67

Table 17: Qwen-397B-A17B vs. Claude Sonnet 5: outcome, calibration error, cost, and effort.

Model	Task(s)	Outcome	Cal. Err.	Cost (USD)	Wall-clock (s)	Steps / TC / Err.
Qwen-397B	`mmlu-astronomy`†	0.715	0.131	0.141	159	19.9 / 25.0 / 2.8
	Pooled GAP_POSITIVE	0.870	0.092	0.315	342	24.7 / 31.4 / 2.6
Claude Sonnet 5	`mmlu-astronomy`†	0.679	−0.043	0.220	210	11.5 / 11.6 / 0.3
	Pooled GAP_POSITIVE	0.828	−0.016	0.407	244	13.5 / 14.2 / 0.5

A.9 Uncertainty of the Axis Effects

The effect size in Table 3 carries no uncertainty on its own. We complement it in two ways. First, we bootstrap the ranking: in each of 1,000 draws, we resample the runs within every cell with replacement (runs that do not complete stay incomplete), recompute all effect sizes, and record which axis ranks first. Second, we fit a linear model to $\mathcal{G}$ of the completed runs of each task, with all five axes as categorical factors (and, for the interactions, all two-way interactions). We report partial η^2, the share of the variance that an axis explains after accounting for the other axes, with 95% intervals from 200 bootstrap draws. *Information* ranks first on every task in at least 99% of the draws, and partial η^2 gives the same order as the effect size, except that *Budget* drops behind *Reasoning*, *Model*, and *Verification* on `redshift-estimation`. Among the two-way interactions, *Information*×*Model* is the largest on `promoter-prediction` ($F = 12.4$, $p < 10^{-12}$, partial $\eta^2 = 0.04$) and `rna-folding` ($F = 32.1$, $p < 10^{-35}$, partial $\eta^2 = 0.13$). *Budget*×*Model* is significant on `rna-folding` ($p < 10^{-3}$) but not on the other two tasks ($p = 0.08$ and $p = 0.36$), and *Information*×*Budget* is significant on `promoter-prediction` ($p = 0.002$), borderline on `redshift-estimation` ($p = 0.055$) and not significant on `rna-folding` ($p = 0.18$).

Table 18: Rank stability and partial η^2 of each axis per task. P(1): share of bootstrap draws in which the axis has the largest effect size. η^2: partial η^2 in a main-effects model, with 95% bootstrap interval.

	`promoter-prediction`		`redshift-estimation`		`rna-folding`	
Axis	P(1)	η^2	P(1)	η^2	P(1)	η^2
Information	1.00	0.413 [0.378, 0.445]	1.00	0.296 [0.281, 0.323]	1.00	0.260 [0.231, 0.289]
Model	0.00	0.080 [0.061, 0.100]	0.00	0.007 [0.002, 0.016]	0.00	0.145 [0.117, 0.177]
Reasoning	0.00	0.011 [0.004, 0.019]	0.00	0.008 [0.002, 0.017]	0.00	0.008 [0.003, 0.018]
Budget	0.00	0.009 [0.004, 0.017]	0.00	0.000 [0.000, 0.004]	0.00	0.004 [0.000, 0.012]
Verification	0.00	0.002 [0.000, 0.008]	0.00	0.003 [0.001, 0.012]	0.00	0.000 [0.000, 0.006]

A.10 Interaction Effects and Non-Completion

Scores are observed only for runs that complete, and completion itself depends on the *Budget*. A drop in the mean score with more budget could therefore come from weaker runs that now complete, rather than from runs becoming worse. We test this in two ways. First, we restrict the comparison to *matched* configurations, i.e., settings of all axes except *Budget* that produce at least k completed runs under every budget level. Second, we report medians next to means. Confidence intervals come from a bootstrap that resamples runs within each cell (1,000 draws). The interactions in Section 4 largely hold under these checks (Table 19). For the small model (Qwen-35B), the mean and median $\mathcal{G}$ still fall with more budget in matched configurations, and for the large model (Qwen-397B) they rise. The drop for the small model disappears only under the strictest matching (at least three completed runs per budget level: mean $\mathcal{G}$ of 0.81 at *short* and 0.82 at *long*), so part of it is a selection effect. At the *interface* level of *Information*, more budget raises both mean and median. At *none* and *identity*, the mean does not change; at *none*, the median rises slightly.

Table 19: Change in $\mathcal{G}$ from the *short* to the *long* budget (long minus short), pooled over the three GAP_POSITIVE tasks, with 95% bootstrap intervals. *Matched* keeps only configurations with at least one completed run under every budget.

	All completed runs		Matched configurations ($k = 1$)	
Level	Mean	Median	Mean	Median
Model = Qwen-35B	−0.067 [−0.093, −0.039]	−0.137 [−0.185, −0.073]	−0.045 [−0.074, −0.017]	−0.107 [−0.157, −0.027]
Model = Qwen-122B	−0.030 [−0.053, −0.010]	−0.018 [−0.033, 0.000]	−0.021 [−0.042, 0.001]	−0.004 [−0.029, 0.005]
Model = Qwen-397B	0.025 [0.000, 0.047]	0.015 [−0.002, 0.027]	0.036 [0.014, 0.059]	0.016 [0.004, 0.028]
Information = none	−0.002 [−0.036, 0.031]	0.057 [0.028, 0.060]	0.004 [−0.027, 0.042]	0.057 [0.027, 0.060]
Information = identity	0.004 [−0.035, 0.040]	0.009 [0.000, 0.078]	0.020 [−0.017, 0.053]	0.012 [0.004, 0.086]
Information = interface	0.045 [0.018, 0.078]	0.028 [0.006, 0.041]	0.045 [0.014, 0.076]	0.031 [0.007, 0.041]
Information = protocol	−0.006 [−0.015, 0.003]	0.000 [0.000, 0.000]	−0.006 [−0.015, 0.003]	0.000 [0.000, 0.000]

A.11 Generation Speed and Tool Time

Because the *Budget* is wall-clock time, a faster model gets more done within the same budget. Table 20 reports, per model, how the wall-clock time of a run splits into LLM generation and tool execution, how many steps a model takes per minute, and how fast it generates. We derive these numbers from the time stamps in the trajectories: the time before an LLM response is counted as generation time, and the time before a tool result as tool time. Tool execution, not generation, takes most of the wall-clock time for every model (54–73%), so the budget mostly limits how much code the agent can run. The three Qwen models take a similar number of steps per minute (6.0–7.2), because the smaller model's faster generation is spent on more and shorter steps; between the *short* and *long* budget, the median number of steps per run grows from 25 to 36 for Qwen-35B and from 18 to 24 for the two larger models. Claude Sonnet 5 takes fewer steps per minute than the open models and spends a larger share of its time executing code, so its stronger sensitivity to the budget is not explained by faster inference. The speed of generation varied only moderately across runs (CV 0.16–0.26), both on our shared model server and through the Anthropic API.

Table 20: Generation speed and use of time per model, medians over all runs of the respective grid. Steps are LLM calls; LLM s / step is the generation time per step. CV is the coefficient of variation of the generation speed across runs.

Model	LLM s / step	Steps / min	Tokens / s	CV	LLM share	Tool share
Qwen-35B	2.4	7.2	127	0.26	30%	70%
Qwen-122B	4.1	6.0	83	0.16	41%	59%
Qwen-397B	4.5	6.1	70	0.24	45%	55%
Step-3.7-Flash	4.3	6.0	94	0.18	46%	54%
Claude Sonnet 5	4.6	3.6	84	0.16	27%	73%

A.12 Use of the Oracle Tool

Since the oracle scores a submission against the test labels, an agent could use it to search for the correct answer instead of verifying its approach. Table 21 shows how often agents called it. Most runs call it rarely, but 8–22% of runs call it five times or more. We flag a run as likely hill-climbing when it calls the oracle at least five times and at least half of the repeated calls follow only file edits, without re-running any code in between. This applies to 4.4% of all oracle runs (210 of 4,800), mostly on `redshift-estimation` (10–15% of its runs), where the most extreme runs call the oracle up to 111 times while tuning the regression step. On `mmlu-astronomy`, where changing single answers would be the most direct form of search, we find almost no such runs (0.1–1.5%). Excluding flagged runs barely changes the pooled $\mathcal{G}$ with the oracle (Qwen-122B: 0.923 to 0.922; Step-3.7-Flash: 0.889 to 0.878), which stays above the baseline without the oracle (0.822 and 0.865).

Table 21: Number of `oracle_check` calls per run (share of runs) and share of runs flagged as likely hill-climbing.

Model	Task	0	1	2	3–4	≥5	Mean calls	Flagged
Qwen-122B	`redshift-estimation`	19.7%	38.9%	8.3%	11.7%	21.4%	4.6	9.7%
	`mmlu-astronomy`	8.1%	68.3%	8.3%	7.5%	7.8%	1.7	0.1%
	`promoter-prediction`	3.8%	44.9%	11.0%	18.9%	21.5%	3.0	2.5%
	`rna-folding`	27.8%	44.6%	9.4%	9.2%	9.0%	1.7	2.8%
Step-3.7-Flash	`redshift-estimation`	44.6%	25.8%	4.8%	5.6%	19.2%	4.0	14.6%
	`mmlu-astronomy`	34.6%	25.2%	12.3%	14.6%	13.3%	2.0	1.5%
	`promoter-prediction`	37.5%	24.0%	7.9%	11.2%	19.4%	2.2	1.0%
	`rna-folding`	49.4%	27.9%	7.9%	5.4%	9.4%	1.9	4.0%

A.13 Axis Analysis for `mmlu-astronomy`

$\mathcal{G}$ is not comparable between the two regimes, so we analyze `mmlu-astronomy` on raw accuracy (Table 22). The correct behavior on this task is to answer with the backbone and decline AstroSage, but agents almost never do this. The model-usage heuristic in Table 7 counts only commands that name the model's path and finds that AstroSage was executed in 66% of runs; if we also count scripts that load AstroSage and are then executed, 99.8% of all runs used it. This explains why accuracy (0.685 on average) stays close to AstroSage's own reference accuracy (0.671) and far below the backbone alone (0.967). Ranked by the same effect size as in Table 3, *Information* (0.93) and *Model* (0.87) matter most, followed by *Budget* (0.15), *Reasoning* (0.08) and *Verification* (0.05). The *protocol* level, which describes how to run AstroSage, lowers accuracy from about 0.71 to 0.62, presumably because it describes the weaker specialist in the most detail and thus points the agent further toward it.

Table 22: `mmlu-astronomy`: completion rate and mean accuracy per level of *Information* and *Model*. Backbone alone: 0.967; AstroSage reference: 0.671; chance: 0.25.

Axis	Level	Completion	Mean accuracy
Information	none	92.8%	0.699
	identity	93.9%	0.712
	interface	98.7%	0.710
	protocol	95.7%	0.617
Model	Qwen-35B	89.6%	0.626
	Qwen-122B	97.5%	0.707
	Qwen-397B	98.8%	0.715

A.14 Sensitivity to the Hurdle Margin

The margin μ decides when a run counts as clearing the hurdle. We set μ per task from the sampling noise of its metric. Table 23 repeats the hurdle statistics with μ set at 0, 0.5, 1, and 2 times its default value per task. The results change little, except in two places. With $\mu = 0$ on `rna-folding`, almost every run clears the hurdle, because its trivial score is 0 and nearly every completed run scores above 0. On `redshift-estimation`, doubling μ (a threshold of $R^2 > 0.5$) removes the weakest hurdle-clearing runs, which raises its ICC. The pooled ICC stays between 45% and 47% for every value.

Table 23: Hurdle failure rate and ICC among hurdle-clearing runs for different multiples of the default margin μ. Pooled: three GAP_POSITIVE tasks, task means removed.

	Hurdle failure rate				ICC, hurdle-clearing			
Task	0μ	0.5μ	1μ	2μ	0μ	0.5μ	1μ	2μ
`redshift-estimation`	48.7%	53.4%	53.8%	54.1%	35.8%	36.2%	36.2%	53.3%
`mmlu-astronomy`	4.2%	4.3%	4.7%	5.5%	6.9%	7.1%	6.9%	7.2%
`promoter-prediction`	0.2%	0.3%	0.5%	0.6%	48.7%	48.7%	49.7%	50.0%
`rna-folding`	0.2%	7.5%	8.7%	9.5%	45.8%	35.7%	35.2%	34.2%
Pooled	16.4%	20.0%	20.6%	20.9%	47.4%	44.8%	46.1%	46.4%

A.15 TAXONOMY RESULTS

Table 24 shows how strongly each axis is associated with the category distribution of each taxonomy dimension, measured by Cramér's V. *Information* ranks first on all six dimensions, consistent with its dominant effect on scores. *Model* and *Budget* show moderate associations, particularly with error categories and results, while *Reasoning* and *Verification* are only weakly associated with every dimension. Figure 8 shows the distribution of categories for each dimension, pooled across all configurations.

Table 24: Association between each axis and the category distribution of each taxonomy dimension in the main experiments, measured by Cramér's V (0 = no association, 1 = perfect association). Axes are ranked within each dimension.

	Error Category		Execution Quality		Model Usage		Planning & Exploration		Result		Verification Behavior	
Axis	Rank	V	Rank	V	Rank	V	Rank	V	Rank	V	Rank	V
Information	1	0.233	1	0.264	1	0.234	1	0.271	1	0.191	1	0.112
Reasoning	5	0.041	4	0.044	4	0.050	4	0.038	4	0.053	4	0.030
Verification	4	0.048	5	0.031	5	0.020	5	0.012	5	0.031	5	0.023
Budget	3	0.120	3	0.070	2	0.108	3	0.039	2	0.152	2	0.108
Model	2	0.157	2	0.138	3	0.097	2	0.084	3	0.095	3	0.067

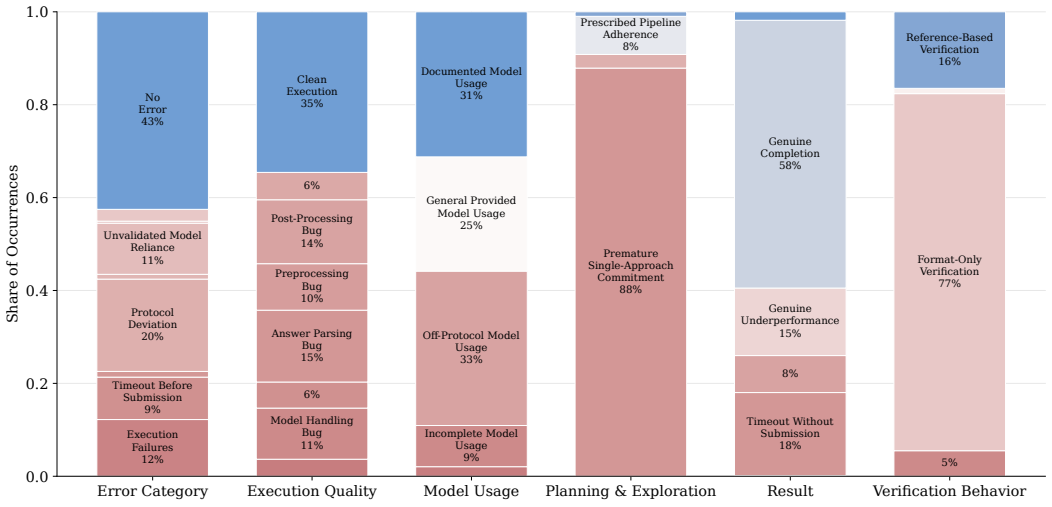


Figure 8: Distribution of categories for each taxonomy dimension in the main experiments.

A.16 AGENT PROMPT

System Prompt for the Agent

```
You are an autonomous research agent. You solve tasks end-to-end, including ones that
call for domain expertise you might not already have. Investigate before you implement,
and ground your methodology in whatever authoritative sources you can find (code,
papers, the web) rather than guessing.

You're given a task description and any data it provides, and one or more pretrained
models that may or may not be relevant to solving it; figuring out whether and how to
use them is part of the task. You have a private workspace to do your own work in. Your
sandbox has GPU access -- check `torch.cuda.is_available()` (or run `nvidia-smi`) to
see what's available, and use it for any GPU-capable workload (e.g. model inference)
rather than defaulting to CPU.

Your context window is limited. Data files can be large (e.g. thousands of CSV rows) --
do NOT read entire large data files with read_file, since its output is truncated and
wastes context. Instead use run_bash with tools like `head`, `wc -l`, or a short
pandas/python snippet to inspect a sample (e.g. the first few rows, column dtypes,
shape) before writing code that processes the whole file.

You have five tools:
- read_file(path): read a file. You may read anything under /task (the task description
and any provided data, read-only), /models (any models mounted for this task, read-only
-- if present, each has its own subdirectory with a README describing what it is --
read it), or /agent_run/workspace (your own scratch directory). Output is truncated for
large files.
- write_file(path, content): write a file. Only paths under /agent_run/workspace are
writable.
- run_bash(command): run a shell command with working directory /agent_run/workspace. A
Python env with numpy, pandas, scikit-learn, scipy, torch, transformers, and accelerate
is already active by default -- just `import` them, no install needed. For anything
beyond that, `uv pip install <package>` adds it to this same env (a shared uv package
cache makes this fast); there is no system `pip` and no writable system site-packages,
so always use `uv pip install`, never plain `pip install`. If you want a fully separate
env instead of extending the default one, `uv venv` + `source .venv/bin/activate` in
the same command (each run_bash call starts fresh, so re-activate every time you need
it). Any model under /models/<name> is a separate matter: it already has its own
complete, pre-built Python environment, entirely unrelated to your workspace's -- do
NOT `uv run`/`uv venv`/`uv pip install` against it or expect it to show up in whatever
environment you build yourself, no matter your current directory. Instead, invoke its
own interpreter directly by absolute path exactly as its README says (typically
`/models/<name>/env/.venv/bin/python your_script.py`) -- that binary already has
everything that model needs installed. Do not use this tool for internet access (e.g.
curl/wget) -- use web_fetch instead.
- web_fetch(url): fetch an http(s) URL and get back its main text content
(ads/nav/boilerplate stripped out), with links kept as [text](url) so you can find
further URLs to fetch. Use this to research a topic, look up library documentation,
check a current fact, or find anything else you need from the internet -- there is no
separate search tool, so to search, fetch a search engine's results page directly, e.g.
https://lite.duckduckgo.com/lite/?q=your+query, then fetch whichever result URL looks
relevant. Some hosts may be blocked; a blocked or failed request returns an error
explaining why instead of stopping the run.
- finish(submission_path, summary, model_used): call this when you are done.
submission_path must be a path under /agent_run/workspace to your final output
(whatever artifact the task asks for -- a predictions file, a report, a script, etc).
summary should briefly describe your approach and findings. If you decide to use one of
the models under /models to produce your output, set model_used to its directory name
(e.g. "astroclip"); if you decided not to use any of them, omit it.
```

```
Work step by step: understand what the task is asking for, gather whatever data or
information you need, do the work before calling finish. You must call finish exactly
once when your final output is ready. Do not stop before calling finish.
```

A.17 Specialist Model and Task Examples

As an example, the following listings show the README of AstroCLIP, the task description of `redshift-estimation`, and the additional information given at each *Information* level of this task. The *none* level adds no information and therefore has no listing.

README for *AstroCLIP*

````
# AstroCLIP

Non-generative cross-modal foundation model for galaxies: a DINOv2 image encoder and a
SpecFormer spectrum encoder, aligned into a shared embedding space via contrastive
pretraining on DESI cross-matched imaging + spectra.

- Model card: https://huggingface.co/polymathic-ai/astroclip
- Paper: https://arxiv.org/abs/2310.03024
- Code: https://github.com/PolymathicAI/AstroCLIP

## Environment

A working environment is already synced at ‘env/.venv‘ -- use it as-is:

‘‘‘bash
/models/astroclip/env/.venv/bin/python your_script.py
‘‘‘

Don’t try to rebuild or reinstall into it -- it’s mounted read-only and already pins
some fussy dependency versions.
````

Task description for `redshift-estimation`

```
## Task

For each galaxy image below, estimate its cosmic redshift. Respond with a single
decimal number per galaxy. Your predictions will be scored against the true redshift
using R^2 (coefficient of determination) across all query galaxies.

## What you have

- ‘data/query_images/<targetid>.npy‘ -- 20 galaxy imaging cutouts, one file per galaxy,
each a ‘(152, 152, 3)‘ float32 array (three photometric bands). No redshift is attached
to these -- that is what you need to estimate.
- ‘data/reference_catalog.npz‘ -- a supporting reference catalog of 126,907 other
galaxies with known redshifts, useful for grounding your estimate.
- One or more pretrained models may be available in your environment under
‘/models/<name>‘ -- each has its own documentation describing what it is and how to use
it. It’s up to you to decide whether any of them are useful for this task.

## Output

Write a CSV file to your workspace with exactly these columns: ‘targetid,redshift‘ --
one row per query galaxy (20 rows total, matching the files in ‘data/query_images/‘).
Then call ‘finish‘ with the path to this file.
```

Information *identity* for `redshift-estimation`

```
## Model identity

The ‘astroclip‘ model mounted at ‘/models/astroclip‘ is the domain-relevant model for
this task.
```

Information *interface* for `redshift-estimation`

````
## Model interface

Load the checkpoint:

‘‘‘python
from astroclip.models import AstroClipModel

model = AstroClipModel.load_from_checkpoint(
    checkpoint_path="/models/astroclip/weights/astroclip.ckpt",
    map_location=device,
    weights_only=False,
)
model.eval()
‘‘‘

Call the model on a batch of preprocessed image tensors to get embeddings:
‘model(tensor, input_type="image")‘ returns a ‘(batch, 1024)‘ embedding. Always embed
in batches of 2 or more -- a batch of exactly 1 crashes on an upstream bug in the
model’s cross-attention head.
````

Information *protocol* for `redshift-estimation`

````
## Protocol

‘data/reference_catalog.npz‘ already holds ‘embeddings‘/‘redshift‘ arrays for 126,907
reference galaxies, pre-embedded exactly the way your query images need to be.

Preprocess raw images before embedding them:

‘‘‘python
from astroclip.data.datamodule import AstroClipCollator

collator = AstroClipCollator()
tensor = collator([{"image": torch.from_numpy(img)} for img in query_images])["image"]
‘‘‘

This applies ‘ToRGB‘, an arcsinh stretch of the raw nanomaggie flux, plus a center-crop
-- required preprocessing, not optional. Skipping it produces near-degenerate
embeddings. This must match how the reference embeddings in
‘data/reference_catalog.npz‘ were produced.

Then estimate redshift with a k-NN regressor:

‘‘‘python
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(reference_embeddings)
reference_scaled = scaler.transform(reference_embeddings)
query_scaled = scaler.transform(query_embeddings)
````

```
knn = KNeighborsRegressor(n_neighbors=64, weights="distance")
knn.fit(reference_scaled, reference_redshift)
predictions = knn.predict(query_scaled)‘‘‘
```

A.18 TAXONOMY PROMPTS

System Prompt for the Taxonomy Judge

```
You are extracting structured information from a log of an autonomous research agent's
attempt at a task, for building a taxonomy of run outcomes. You will be shown the task
description, the complete raw event log (trace.jsonl -- one JSON object per line:
run_start, llm_response, tool_call, status_message, run_end, in chronological order),
mechanical facts computed from the trace and the task's scoring, the task's intended
approach, and the agent's final workspace. Read everything before answering; do not
guess from a partial read. Some scoring facts are the agent's own self-reported claims,
marked as such below -- treat those as claims to confirm or contradict using the trace,
not as established fact. Respond with a single JSON object matching exactly the fields
requested, and nothing else -- no markdown fencing, no commentary outside the JSON
object. Make sure the JSON is syntactically valid: escape any quotation marks that
appear inside a string value.
```

Prompt for the Taxonomy's *Result* Dimension

```
Describe, in your own words, how this run concluded and whether the result holds up,
drawing on the scoring facts above. A clean, genuine success; a result that looks gamed
or exploited somehow (hardcoded/memorized answers instead of a real method, peeking at
solution or ground-truth files, exploiting a metric loophole, a suspiciously trivial
method for a result this good); a crash or a submission that never became genuinely
scoreable, and specifically what happened; or something else entirely. Say plainly if
there's nothing questionable about it.
```

Prompt for the Taxonomy's *Error Category* Dimension

```
Name the single root cause of any problem with this run, not every symptom -- a
cascading failure usually traces back to one earlier cause, and downstream symptoms of
that same cause don't each count separately. Distinguish a genuine error from something
that only looks like one: normal exploration, a failed attempt the agent recognized and
moved past, or a hypothesis it later revised are not errors on their own. If there's no
error attributable to a specific cause -- including for a run that succeeded cleanly --
say so explicitly with "no_identifiable_error"; don't invent a cause to fill the field.
```

Prompt for the Taxonomy's *Specialist Model Usage* Dimension

```
How did the agent engage with the domain-specific pretrained model(s) available for
this task? Describe, in your own words, specifically what happened -- name the model(s)
involved and what was actually done with them (or not done). Draw on the mechanical
engagement signals and the agent's own self-reported model choice given above,
alongside the trace itself. A few examples of the kind of thing to look for, not a list
to pick from -- if what you observe is something else entirely, describe that instead:
reading documentation before using a model correctly; using one correctly without
apparently reading its docs first; loading a model but using it incorrectly in some
specific way (say exactly how -- wrong input format, wrong preprocessing, misreading
its output, using an untrained/randomly-initialized part of it, applying it to the
```

```
wrong data); never loading or running it at all; using a different, unintended model
instead.
```

Prompt for the Taxonomy's *Verification Behavior* Dimension

```
What did the agent actually do, if anything, to check its own work before submitting?
Describe, in your own words, the specific check performed (or the absence of one) --
not just whether it was rigorous, but what it actually consisted of. This is about
observed behavior, independent of what the task's own Verification condition nominally
asked for or permitted.
```

Prompt for the Taxonomy's *Planning and Exploration* Dimension

```
Describe the agent's overall approach and how it arrived at it, in your own words. Did
it explore more than one substantively different approach before committing, or settle
on one early and stick with it even where the trace shows evidence (an error, an
unexpected result) that should have prompted reconsidering it? Name the specific
approach(es) involved, not just whether it explored.
```

Prompt for the Taxonomy's *Execution Quality* Dimension

```
Independent of whether the agent's plan was sound, was the plan implemented correctly?
The agent's actual code is included above in its final workspace -- read it directly,
not just the trace's account of it, to find concrete bugs. Describe, in your own words,
any concrete implementation bugs found -- wrong preprocessing, off-by-one errors,
misread columns or fields, output-format mismatches, incorrect use of an API or library
-- or state plainly that none were found.
```