AGENTS-EVAL: A LIFECYCLE PLATFORM FOR END-TO-END AGENT EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents are increasingly used for complex tasks in software engineering, information seeking, and scientific research. Evaluating these agents is essential for reliable comparison and identifying where further improvement is needed. Yet comparing agents requires aligning benchmark protocols, agent harnesses, and execution environments. Researchers often repeat this benchmark-specific integration and configuration, incurring substantial engineering effort while leaving evaluation conditions inconsistent across studies. Existing evaluations also rely largely on aggregate scores and broad task categories, offering limited insight into which capabilities tasks require and at what demand levels. We introduce AGENTS-EVAL, an open-source platform that addresses these two challenges. First, the platform provides reusable benchmark integrations and a unified evaluation workflow, standardizing execution conditions and lifecycle management while preserving benchmark-native interaction and scoring semantics. This allows researchers to reuse maintained evaluation setups across models and harnesses. Second, it introduces a capability translation matrix that decomposes heterogeneous tasks into six shared capability dimensions and assigns an explicit capability demand level to each. These annotations describe task requirements independently of agent responses and support comparisons across demand levels, models, and harnesses. Our experiments reveal shared capability demands across benchmark domains, harness gains that vary with model choice and task requirements, and recurring environment, service, and verification failures. Agents-Eval combines reusable evaluation workflows with fine-grained capability analysis to guide improvements in models, harnesses, and evaluation infrastructure.

1 INTRODUCTION

Evaluating language-model agents is essential for understanding and improving their performance on complex tasks, including software engineering, information seeking, and scientific research Jimenez et al. (2024); Wei et al. (2025); Xu et al. (2026b). Such evaluations should support reliable comparisons and provide actionable evidence for further development. However, the growing number of reported benchmark results has not been accompanied by equally consistent evaluation practices Kapoor et al. (2026); **?**. Studies often differ in their agent configurations, execution environments, and reporting protocols, while aggregate scores provide limited insight into the sources of performance differences. Consequently, it remains difficult to determine whether a reported improvement reflects stronger model capabilities, a more effective agent harness, or differences in evaluation conditions, and which component should be improved next.

The first challenge is aligning evaluation conditions across models, harnesses, and benchmarks for reproducible comparison. Unlike isolated model responses, agent trajectories emerge from interactions among the underlying model, the harness that manages its execution, and a mutable environment. Context management, tool interfaces, resource budgets, software dependencies, network availability, and verifier behavior can all affect the final outcome Wang et al. (2026c); Ni et al. (2026); **?**. Researchers often repeat benchmark-specific integration and environment preparation, incurring substantial engineering effort while leaving evaluation conditions inconsistent across studies. When these conditions are insufficiently controlled or documented, evaluations conducted under the same benchmark name may measure different effective systems. Discrepancies between originally reported and independently obtained results can therefore be difficult to interpret. Recent

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.

infrastructures, including AgentCompass Ding et al. (2026) and the Holistic Agent Leaderboard (HAL) Kapoor et al. (2026), have made substantial progress in standardizing evaluation interfaces and execution workflows. Building on these advances, reproducible comparison requires explicit control and traceability across the entire evaluation lifecycle, including execution failures and verification.

The second challenge is the diagnostic resolution of existing evaluations. Realistic benchmark tasks typically require a combination of capabilities, whereas results are commonly summarized by benchmark-level scores or broad domain labels. A scientific research task, for example, may require external information search, research planning, code implementation, tool use, and scientific validation within a single trajectory Xu et al. (2026b). Failure on such a task does not, by itself, identify which requirement was difficult to satisfy. Moreover, benchmarks from different domains may share substantial capability demands, while tasks within the same benchmark may require different combinations of capabilities. Without an explicit decomposition into capability dimensions and their demand levels, aggregate comparisons offer limited guidance for targeted model and harness improvements Zhou et al. (2026a). Addressing this challenge requires connecting task requirements with observed outcomes and execution evidence.

We introduce Agents-Eval, a unified platform for reproducible and diagnostic agent evaluation. It brings benchmark integration, agent execution, and environment management into a common workflow while preserving native evaluation semantics. Lifecycle-wide control and traceable execution evidence help distinguish unsuccessful task completion from failures in the evaluation process, providing a more reliable basis for comparing models and harnesses.

To support targeted diagnosis, Agents-Eval constructs a capability translation matrix that maps heterogeneous task requirements onto six shared dimensions and their demand levels. Relating these requirements to evaluation outcomes reveals differences concealed by aggregate scores and broad benchmark categories. Together with execution evidence, this representation helps identify where model or harness improvements are needed.

Using Agents-Eval, we demonstrate capability translation on six representative benchmarks, illustrating how a shared capability space characterizes heterogeneous task requirements across domains. We further compare model–harness configurations and analyze execution failures to investigate how task demands, harness choices, and evaluation conditions shape agent performance. Our analysis reveals substantial overlap and variation in capability demands that broad benchmark categories do not capture. It also identifies recurring environment, service, and verification failures, highlighting the importance of examining execution evidence when interpreting unsuccessful attempts. Furthermore, the relative performance of harnesses varies across underlying models and task-demand groups. These results identify the relationship among the model, harness, and task requirements as an important consideration in agent evaluation: harness effectiveness cannot be adequately characterized by a single aggregate improvement.

Our contributions are twofold:

- A unified platform for reusable and reproducible agent evaluation. We release Agents-Eval, which integrates benchmarks, harnesses, and managed environments through a common evaluation workflow. Reusable benchmark integrations and consistent lifecycle management reduce repeated setup and alignment effort while preserving native interaction and scoring semantics.
- Capability translation for fine-grained evaluation and diagnosis. We introduce a capability translation matrix that decomposes tasks into six shared capability dimensions and explicit capability demand levels. By relating these annotations to outcomes and execution evidence, our experiments characterize benchmark requirements, reveal model-dependent harness gains, and identify recurring failure sources to guide targeted improvements.

2 Related Work

Benchmarks and diagnostics. Recent benchmarks extend agent evaluation to professional workflows Xu et al. (2026a); Patwardhan et al. (2026); Jha et al. (2025), long-context interactions Li et al. (2026), non-ideal execution conditions Xi et al. (2026), and heterogeneous tool ecosys-

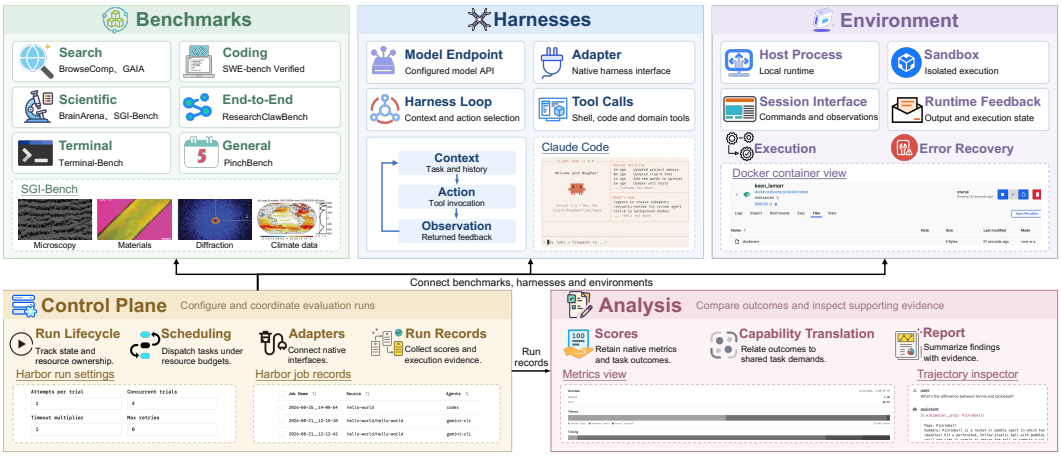


Figure 1: Overview of Agents-Eval. The platform connects benchmark suites, agent harnesses, and managed execution environments through a control plane, and links each run to scores, capability demand profiles, evidence, and analysis.

tems Guo et al. (2026). These settings combine multiple capability requirements, limiting the diagnostic value of aggregate scores. AgentIF examines instruction following under complex constraints Qi et al. (2025), Plan-RewardBench evaluates planning and error recovery Wang et al. (2026a), and DREAM assesses factual and temporal validity in research outputs Ben Avraham et al. (2026). Task-setup and reward-design errors can also distort reported performance Zhu et al. (2026). Following the general-scale approach Zhou et al. (2026a), AGENTS-EVAL decomposes heterogeneous tasks into six shared capability dimensions and relates outcomes to their demand levels while preserving native evaluation semantics.

Evaluation infrastructure. HAL standardizes model–harness configurations and execution backends for cost-aware evaluation Kapoor et al. (2026), while AgentCompass separates benchmarks, harnesses, and environments into reusable components Ding et al. (2026). ResearchEnvBench and GitTaskBench nevertheless identify environment setup, dependencies, and runtime compatibility as persistent sources of evaluation failure Wang et al. (2026c); Ni et al. (2026). Agent reliability also requires assessing consistency, robustness, predictability, and failure severity beyond aggregate accuracy Rabanser et al. (2026). AGENTS-EVAL combines reusable benchmark integrations with lifecycle management, including sandbox scheduling, fault handling, and resource reclamation. Linking execution evidence with capability demand profiles supports both reproducible comparison and investigation of performance differences.

3 AGENTS-EVAL

3.1 DESIGN OBJECTIVES AND SYSTEM OVERVIEW

Agents-Eval is a platform for consistent execution, traceable outcomes, and capability-based diagnosis. Its control plane is the component responsible for evaluation scheduling and lifecycle management. A common interface alone is insufficient for standardized evaluation because reproducible agent comparison requires lifecycle-wide control over the concrete execution environment. The platform therefore maintains benchmark-specific execution requirements and connects task demands with scores and execution evidence to support targeted model and harness improvement.

Agents-Eval models an evaluation attempt as the interaction of a benchmark, an agent, and a managed execution environment under a unified control plane. Figure 1 summarizes the architecture. The control plane connects benchmark preparation, agent execution, verification, and resource reclamation into a single managed lifecycle. The lifecycle produces a unified results and diagnosis layer that associates scores with trajectories, logs, artifacts, capability demand profiles, failure attribution, and the final analysis report.

Table 1: Representative agent benchmark entries currently integrated in Agents-Eval. Multiple harnesses on one row are alternatives the platform maintains for that benchmark, listed with the platform default first.

Task domain	Benchmark	Agent harness
End-to-End Research	ResearchClawBench Xu et al. (2026b)	Claude Code, ResearchHarness, Codex
	SGI-Bench Xu et al. (2025)	ResearchHarness
General Agent Tasks	PinchBench	OpenClaw
Professional Workflows	ALE-CLI Sun et al. (2026)	Claude Code, Codex, ALE-claw
Terminal Coding	TerminalBench Merrill et al. (2026)	Terminus-2, mini-SWE-agent, Claude code
Agentic Coding	SWE-bench Verified Jimenez et al. (2024)	mini-SWE-agent, SWE-agent, OpenHands, Claude Code
	SWE-bench Multilingual Zan et al. (2026)	mini-SWE-agent, SWE-agent, OpenHands, Claude Code
	SWE-bench Pro Deng et al. (2025)	mini-SWE-agent, SWE-agent, OpenHands, Claude Code
Scientific Reasoning	BrainArena Wu et al. (2026)	Claude Code, SeekBrain Wu et al. (2026)
	BiomniBench Qu et al. (2026)	Claude Code
Web Search	BrowseComp Wei et al. (2025)	Agents-A1-Search Bai et al. (2026)
	BrowseComp-ZH Zhou et al. (2025)	Agents-A1-Search
	GAIA Mialon et al. (2024)	Agents-A1-Search
	Seal-0 Pham et al. (2026)	Agents-A1-Search

3.2 System Components and Evaluation Contract

Agents-Eval organizes each evaluation around five components: *Benchmark*, *Agent*, *Environment*, *Control Plane*, and *Analysis*. *Benchmarks* define task suites and provide the assets, evaluators, and execution protocols used by the platform. Agents-Eval represents each benchmark through a unified specification of its task instances, interaction protocol, and native evaluator. This specification preserves the benchmark's task requirements and scoring criteria when evaluating different agents. Table 1 summarizes the supported benchmarks and agent harnesses.

The platform attaches capability annotations through a shared rubric pipeline, while each benchmark remains a structured evaluation entry that can be selected and executed through the same control-plane interface. *Agents* combine model endpoints with harnesses that govern context construction, tool or action calls, state management, and termination, while preserving their interaction behavior.

Sandbox environments provide the concrete state and resources in which agents act. They expose a common execution session while retaining pinned data, dependencies, resource limits, and network conditions. The *control plane* coordinates these components, owns each evaluation throughout execution and reclamation, and links live events and artifacts to its result record.

The *analysis* layer converts these records into task and aggregate scores, capability demand profiles, analysis reports, and inputs for human review. The contract therefore standardizes execution responsibility and evidence continuity while preserving benchmark-native interaction and scoring semantics. It does not impose a universal task protocol or make scores from different benchmarks directly comparable.

3.3 Controlled Execution and Lifecycle Management

Agent evaluation involves interaction with an execution environment, where agents run commands, manipulate files, and use tools to observe and modify the task state. Their outcomes therefore depend on environment images, installed dependencies, initial state, resource limits, and network access. Existing frameworks provide modular interfaces and reusable runtimes for heterogeneous benchmarks Ma et al. (2024); Le Sellier De Chezelles et al. (2025); Ding et al. (2026), but benchmark-

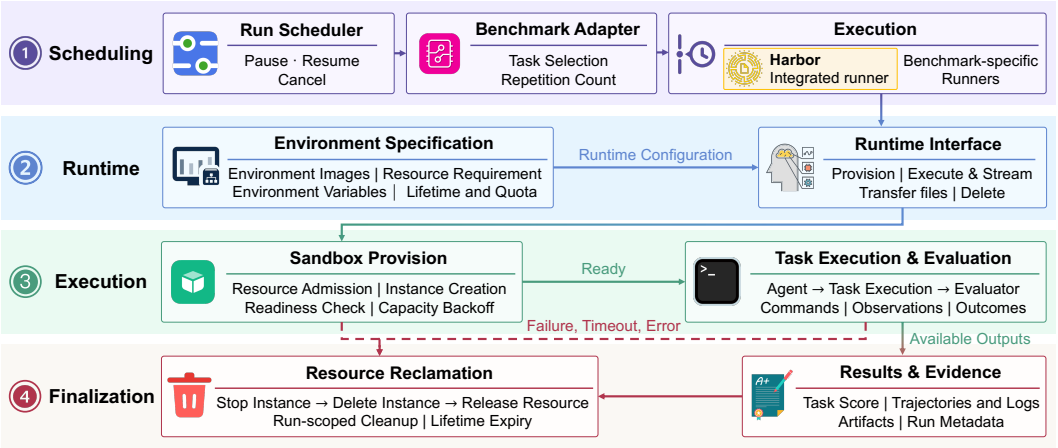


Figure 2: Evaluation lifecycle in Agents-Eval, from task scheduling and environment preparation to execution and finalization. The control plane manages normal completion and exceptional paths, preserving available evidence and reclaiming resources.

specific requirements still need to be configured and maintained throughout execution. Figure 2 summarizes how Agents-Eval manages these requirements across the evaluation lifecycle.

The control plane coordinates benchmark adapters and supported runners, such as Harbor, through a shared runtime interface. It schedules tasks according to resource availability, provisions the required sandbox, and verifies that the specified environment is ready before execution. It then launches and monitors the agent harness in the prepared environment, collecting trajectories, logs, and generated artifacts, and coordinates recovery from execution interruptions while preserving available results and evidence. The platform invokes the benchmark-native evaluator to assess task outputs and retains its results with the execution record. After completion, interruption, or cancellation, Agents-Eval coordinates termination and resource reclamation across runtime backends. A persistent run identifier links the benchmark specification, agent configuration, execution conditions, trajectories, evaluator output, and cleanup status. These records help researchers distinguish evaluation failures from unsuccessful task attempts and investigate discrepancies across evaluations.

3.4 Capability Demand Profiling and Analysis

An aggregate benchmark score can conceal an agent's strengths and weaknesses, offering little guidance on what to improve. Agents with similar overall scores may succeed on tasks that require very different capabilities. To support targeted optimization, Agents-Eval characterizes each task by the capabilities it requires and the demand level of each capability, following the general-scale approach Zhou et al. (2026a). Comparing outcomes across demand levels lets us test whether and how performance changes with each capability requirement, helping identify priorities for model and harness improvement.

Our dimensions reflect capability areas emphasized in model evaluations Qwen Team (2026); DeepSeek-AI (2026); Anthropic (2026); OpenAI (2026), including instruction following, tool use, scientific reasoning, and coding. Research agents additionally require iterative planning, information search, and evidence synthesis OpenAI (2026). We organize these requirements into six capability dimensions: **Instruction and Constraint Adherence (ICA)**, the demand for satisfying explicit instructions and interacting constraints, **Tool Complexity (TC)**, the complexity of selecting, using, and coordinating tools, **External Information Search and Synthesis (EISS)**, the demand for retrieving and integrating external evidence, **Research Lifecycle Planning and Execution (RLPE)**, the demand for organizing and adapting a research process, **Scientific Reasoning and Validation (SRV)**, the depth of scientific reasoning and evidence-based validation, and **Code Implementation and Execution (CIE)**, the difficulty of implementing, debugging, and validating executable code. We use `gpt-5.6-sol` to assign a *capability demand level* from 0 (no demand) to 5 (very high demand) to each dimension separately, using the rubrics in Appendix A.6. A demand level describes a

task requirement, not the evaluated agent's proficiency; a task may place high demands on multiple dimensions.

The six annotations form a task vector $x_t \in \{0, \ldots, 5\}^6$. Stacking these vectors for a benchmark with n tasks yields its capability translation matrix $X_b \in \{0, \ldots, 5\}^{n \times 6}$. Combined with task outcomes, this matrix supports comparisons of agent performance at different demand levels within each capability, providing a basis for identifying strengths and potential improvement areas beyond aggregate scores. In Section 4.2, we compare capability demand profiles across public benchmarks and examine how outcomes vary with demand level.

4 EXPERIMENTS

The experiments use Agents-Eval to ask what can be learned when agent evaluation is organized around task demands, harness choices, and lifecycle evidence rather than a single aggregate score. We keep the evaluation protocol and lifecycle control explicit so that the resulting comparisons reflect agent behavior instead of untracked execution variation. We organize the study around three research questions:

- **RQ1.** How does capability translation characterize heterogeneous benchmark demands?
- **RQ2.** How do harness gains and recoverable performance vary across capability demands?
- **RQ3.** What undermines reliable agent evaluation?

Together, these questions characterize benchmark demands through a shared capability space, examine whether harness gains generalize across capabilities and models, and identify recurring threats to evaluation reliability. The findings inform both agent improvement and the design of reliable evaluations.

4.1 EXPERIMENTAL SETUP

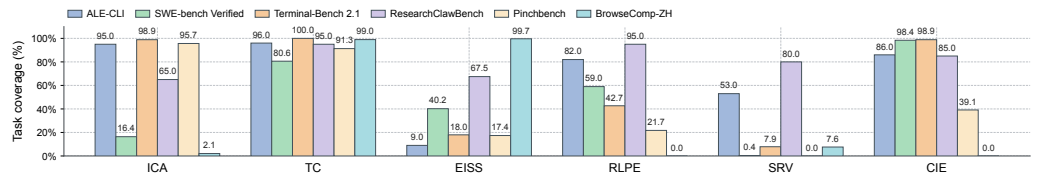
Configurations. All experiments are conducted through Agents-Eval. We select representative model endpoints and evaluate them with the compatible agent harnesses maintained for each benchmark, allowing us to compare both model and harness effects. All benchmark splits, agent configurations, execution environments, resource limits, and native evaluators follow their official settings. Additional implementation details are provided in Appendix A.3.

Metrics. We retain each benchmark's native task-level outcome. Capability coverage is the fraction of tasks with a nonzero capability demand level. Mean demand level averages the 0–5 annotations across all tasks in a benchmark. For comparisons within each capability dimension, tasks at demand level 0 are excluded; the remaining tasks are grouped as Low (demand levels 1–2), Medium (3), and High (4–5). Binary outcomes are summarized by task success rate with 95% Wilson confidence intervals, while graded outcomes use the benchmark-native mean score. Groups containing fewer than 20 tasks are shown descriptively but are not used as primary evidence.

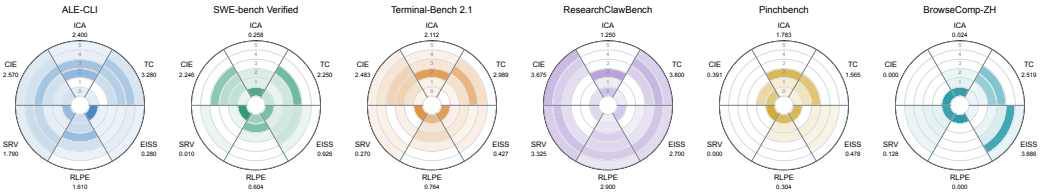
4.2 RQ1: HOW DOES CAPABILITY TRANSLATION CHARACTERIZE BENCHMARK DEMANDS?

Benchmark capability demand profiles. The translation matrix characterizes benchmarks beyond broad labels such as coding or general agent tasks. Figure 3 compares (a) capability coverage and (b) mean demand levels across six benchmarks. CIE is required by nearly all SWE-bench Verified and Terminal-Bench 2.1 tasks, consistent with their established coding focus in the agent evaluation community. PinchBench primarily requires ICA and TC. Compared with these benchmarks, ALE-CLI combines broader research planning and scientific reasoning requirements with instruction, tool, and coding demands. ResearchClawBench exhibits particularly high CIE demand alongside substantial RLPE and SRV demand. The profiles thus recover established benchmark specializations while revealing shared capability requirements and differences in demand level that broad domain labels conceal. This agreement supports the interpretability of capability translation and its use in benchmark selection and result analysis.

Demand sensitivity. We examine how success rates change as capability demand increases across models and dimensions. With Claude Code fixed, we pool tasks from SWE-bench Verified,



(a) Capability coverage



(b) Mean capability demand level

Figure 3: Capability translation across six benchmarks. (a) Percentage of tasks with nonzero demand in each capability dimension. (b) Mean capability demand level on the 0–5 rubric for each benchmark and dimension. These profiles characterize task requirements rather than agent performance.

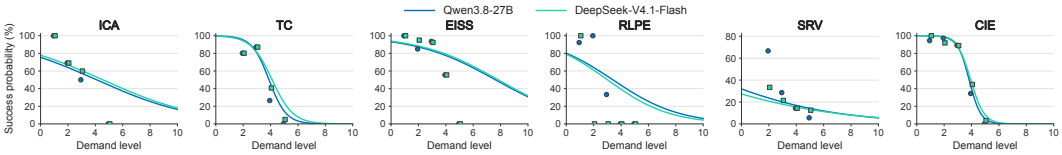


Figure 4: Observed success rates (markers) and weighted logistic fits (curves) by capability demand level for two models under Claude Code. Only demand levels 1–5 contain observed tasks; curves beyond demand level 5 are extrapolations.

Terminal-Bench 2.1, ALE-CLI, and ResearchClawBench. For each model m and capability k, we retain tasks where k is tied for the highest demand level and fit $\hat{p}_{m,k}(\ell) = \sigma(\alpha_{m,k} + \beta_{m,k}\ell)$, where $\ell \in \{1, \ldots, 5\}$ is the demand level and σ is the logistic function. Following the fitting procedure of Zhou et al. (2026a), sparsely populated demand levels receive less weight, and an artificial zero-success anchor at demand level 20 stabilizes extrapolation. Figure 4 compares observed rates and fitted curves for Qwen3.8-27B and DeepSeek-V4.1-Flash. The decline is clearest for TC and CIE; sparse observations at high demand levels limit conclusions for the other dimensions.

4.3 RQ2: How Do Harness Gains and Recoverable Performance Vary Across Capability Demands?

Harness gains. Agent benchmarks often have a companion harness, such as ALE-claw for ALE-CLI Sun et al. (2026). Yet a benchmark score reflects both the model and the harness that governs its interaction with the task. We first test whether changing the harness materially changes outcomes while holding the model, tasks, evaluator, and environment fixed. We then examine how the resulting gains vary with task capability demands and whether they persist across models.

Specifically, we compare Claude Code, OpenHands, and mini-SWE-agent on SWE-bench Verified; Claude Code, Terminus-2, and mini-SWE-agent on Terminal-Bench 2.1; and Claude Code, ALE-Claw, and Codex on ALE-CLI. For each benchmark and model endpoint, we compare harnesses within capability-demand groups while holding the task set, evaluator, and environment fixed. Figure 5 reports task accuracy for SWE-bench Verified and Terminal-Bench 2.1, and mean task score for ALE-CLI. Within each benchmark and model, differences among harnesses indicate the harness gain. Comparing these differences across model endpoints reveals whether gains generalize beyond one model or arise from a specific model–harness pairing. Translucent bars mark groups with fewer than 20 tasks and are retained for completeness rather than used as primary evidence.

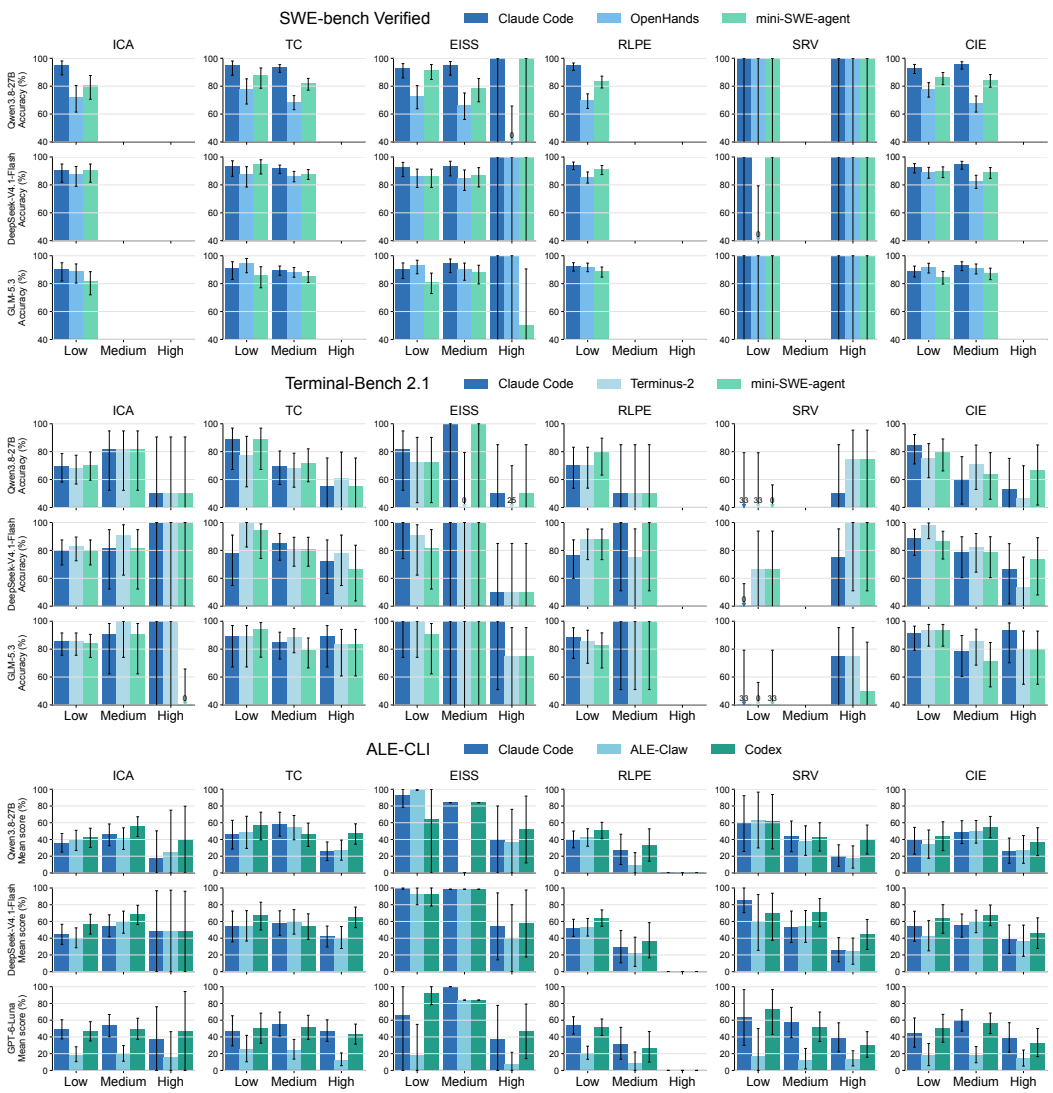
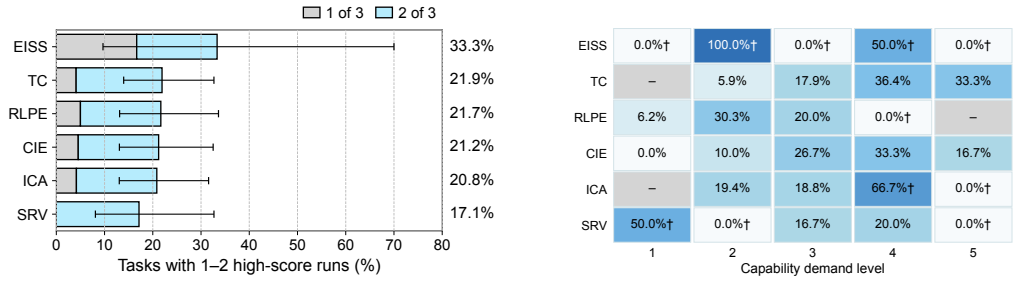


Figure 5: Harness outcomes on SWE-bench Verified, Terminal-Bench 2.1, and ALE-CLI, stratified by model, capability dimension, and demand group. The first two benchmarks report task accuracy; ALE-CLI reports mean task score. Within each benchmark and model, differences compare harness choices. Error bars show uncertainty across tasks; translucent bars indicate groups with fewer than 20 tasks.

Recoverable performance. Beyond comparing harnesses, repeated runs under a fixed configuration reveal where high-score performance is attainable but not yet consistent. We retain tasks with three valid scores and nonzero demand in the relevant capability, allowing tasks to belong to multiple groups. Figure 6(a) shows intermittent high scores across all six capability groups. The breakdown by demand level exposes differences hidden by these aggregates: the intermittent share reaches 36.4% for TC at demand level 4 and 33.3% for CIE at demand level 4, compared with 30.3% for RLPE at demand level 2. Thus, opportunities to stabilize demonstrated performance occur at different demand levels across capabilities. These groups provide concrete targets for trajectory diagnosis and subsequent harness refinement. The observations identify improvement candidates rather than establish learning gains or attribute the variation to the harness; sparse groups, particularly EISS, also preclude firm comparisons. Task-level examples are shown in Appendix Figure 9.



(a) By capability **(b)** By demand level

Figure 6: Intermittent high scores on ALE-CLI with Qwen3.8-27B: tasks scoring ≥ 0.8 in exactly one or two of three runs, grouped by (a) capability and (b) demand level. Error bars: 95% Wilson intervals. †: fewer than five tasks; dash: no tasks.

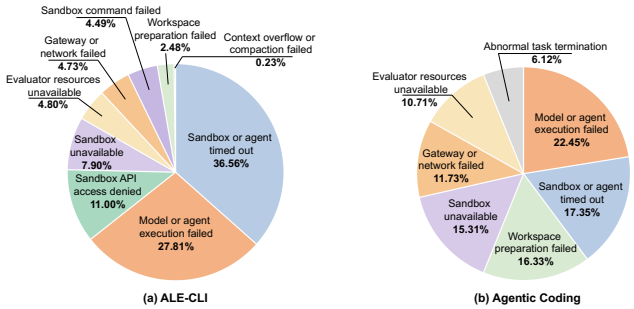


Figure 7: Distribution of recorded failure categories in (a) ALE-CLI and (b) coding evaluations. Percentages are computed within the analyzed failure records for each panel, rather than over all evaluated tasks.

4.4 RQ3: What Undermines Reliable Agent Evaluation?

We analyze failed ALE-CLI and coding runs using lifecycle events, execution traces, and evaluator outputs. Figure 7 shows recurring sandbox, service, and evaluation-pipeline failures alongside model or agent execution failures. Although their frequencies differ, these shared categories suggest that reliability safeguards can be reused across benchmarks. Percentages describe the analyzed failures, not all evaluated tasks.

Reliable evaluation therefore requires distinguishing unsuccessful task completion from failures of the evaluation process. Agents-Eval combines environment checks, selective retries, and retained evidence to support this distinction. Recovery should address transient infrastructure faults without masking task-level failures. Appendix Table 2 maps the observed categories to these controls, making fault handling an explicit part of the evaluation protocol.

5 Conclusion

We presented Agents-Eval, an open-source platform combining lifecycle control and capability translation for reproducible and diagnostic agent evaluation. By connecting task demands, benchmark scores, and execution evidence, it supports analysis beyond aggregate results. Our experiments reveal distinct benchmark capability demand profiles, harness gains that vary across models and task demands, and recurring infrastructure failures. These findings inform targeted agent improvement and reliable evaluation design. The current scope of Agents-Eval is limited to evaluation and diagnosis. A natural next step is to connect capability analysis with targeted training and reevaluation, assessing whether the identified gaps can guide measurable improvements.

AI USE STATEMENT

We study the evaluation of large language models and LLM-based agents. We use an LLM to assign capability demand levels to tasks according to our rubric, as described in the methodology. We also use AI coding tools to assist with developing the Agents-Eval web interface and adapting the platform to different computing resources and execution environments.

We use LLM tools to assist with surveying prior work, polishing the manuscript's language, and adjusting its LaTeX formatting, including table layout and row spacing. We do **not** use these tools to determine the core research idea and experimental design. We supply all numerical values, experimental results, analyses, and citations for manuscript preparation and verify them against actual experimental records and original sources. We do **not** use AI-generated statements as substitutes for experimental evidence or cited sources. We review the final text and take full responsibility for the paper.

ETHICS STATEMENT

This work studies LLM-based agent evaluation using existing benchmarks and does not involve human-subject research. Agents-Eval supports controlled execution and retention of evaluation evidence to facilitate inspection of agent behavior and evaluation failures. The reported results characterize performance under the tested configurations and should not be interpreted as guarantees of safe or reliable deployment.

REPRODUCIBILITY STATEMENT

We open-source Agents-Eval at `https://anonymous.4open.science/r/Agents-Eval-ICLR/`, including the platform interface, benchmark evaluation code, lifecycle scheduling, and fault-handling mechanisms. The platform can be deployed on developer-managed clusters or servers by configuring the model endpoints, CPU and memory allocations, storage, and sandbox backend for the available infrastructure through the provided templates and runtime interfaces. Deployment instructions are included in the repository, while experimental settings and capability annotation rubrics are detailed in Appendices A.3 and A.6, respectively.

REFERENCES

Anthropic. Introducing Claude Opus 5, 2026. URL `https://www.anthropic.com/news/claude-opus-5`.

Lei Bai, Zongsheng Cao, Yang Chen, Zhiyao Cui, Shangheng Du, Yue Fan, Shiyang Feng, Zijie Guo, Haonan He, Liang He, et al. Scaling the horizon, not the parameters: Reaching trillion-parameter performance with a 35b agent. *arXiv preprint arXiv:2606.30616*, 2026.

Elad Ben Avraham, ChangHao Li, Ron Dorfman, Roy Ganz, Oren Nuriel, Amir Dudai, Aviad Aberdam, Noah Flynn, Elman Mansimov, Aditya Kalyanpur, and Ron Litman. DREAM: Deep research evaluation with agentic metrics. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9879–9904. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.448.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026. URL `https://arxiv.org/abs/2609.19969`.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Zichen Ding, Jiaye Ge, Shufan Jiang, Kai Chen, Mo Li, Qingqiu Li, Zehao Li, Zonglin Li, Tiaohao Liang, Shudong Liu, et al. Agentcompass: A unified evaluation infrastructure for agent capabilities. *arXiv preprint arXiv:2607.13705*, 2026.

Zikang Guo, Benfeng Xu, Chiwei Zhu, Wentao Hong, Xiaorui Wang, and Zhendong Mao. MCP-AgentBench: Evaluating real-world language agent performance with MCP-mediated tools. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30888–30896, 2026. doi: 10.1609/aaai.v40i37.40347.

Saurabh Jha, Rohan R. Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, et al. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 27134–27197. PMLR, 2025. URL https://proceedings.mlr.press/v267/jha25a.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Sebastian Joseph, Syed M Husain, Stella Offner, Stéphanie Juneau, Paul Torrey, Adam Bolton, Juan Farias, Niall Gaffney, Greg Durrett, and Junyi Jessy Li. Astrovisbench: A code benchmark for scientific computing and visualization in astronomy. *Advances in Neural Information Processing Systems*, 38, 2026.

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 *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, et al. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025.

Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, et al. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7422–7440. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.337.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. In *Advances in Neural Information Processing Systems*, 2024.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Ziyi Ni, Huacan Wang, Shuo Zhang, Shuo Lu, Ziyang He, Zhenheng Tang, Sen Hu, Bo Li, Chen Hu, Binxing Jiao, et al. Gittaskbench: A benchmark for code agents solving real-world tasks through code repository leveraging. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32564–32572, 2026.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026. URL https://openai.com/index/gpt-6-astra/.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simon Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. In *International Conference on Learning Representations*, volume 2026, pp. 24005–24040, 2026.

Thinh Pham, Nguyen Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. Sealqa: Raising the bar for reasoning in search-augmented language models. In *International Conference on Learning Representations*, volume 2026, pp. 98989–99017, 2026.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. AgentIF: Benchmarking large language models instruction following ability in agentic scenarios. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1892.

Yuanhao Qu, Yingzhou Lu, Xinming Tu, Serena Zhang, Tianwei She, Alexander Glenn Shaw, Jou-Ho Shih, Bingqing Zhao, Minjie Shen, Haochen Yang, et al. Biomnibench: Process-level evaluation of llm agents for real-world biomedical research. *bioRxiv*, pp. 2026–05, 2026.

Qwen Team. Qwen3.8-2.4T-A95B model card, 2026. URL `https://huggingface.co/Qwen/Qwen3.8-2.4T-A95B`.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL `https://arxiv.org/abs/2602.16666`.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, et al. Agents' last exam. *arXiv preprint arXiv:2606.05405*, 2026.

Minyang Tian, Luyu Gao, Shizhuo D Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, et al. Scicode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems*, 37:30624–30650, 2024.

Jiaxuan Wang, Yulan Hu, Wenjin Yang, Zheng Pan, Xin Li, and Lan-Zhe Guo. Aligning agents via planning: A benchmark for trajectory-level reward modeling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23174–23200. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.1062.

Weida Wang, Dongchen Huang, Jiatong Li, Tengchao Yang, Ziyang Zheng, Chuyi Peng, Di Zhang, Dong Han, Benteng Chen, Binzhao Luo, et al. Cmphysbench: A benchmark for evaluating large language models in condensed matter physics. In *International Conference on Learning Representations*, volume 2026, pp. 3388–3421, 2026b.

Yubang Wang, Chenxi Zhang, Bowen Chen, Zezheng Huai, Zihao Dai, Xinchi Chen, Yuxin Wang, Yining Zheng, Jingjing Gong, and Xipeng Qiu. Researchenvbench: Benchmarking agents on environment synthesis for research code execution. *arXiv preprint arXiv:2603.06739*, 2026c.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Jiamin Wu, Peishan Xiang, Jingyang Chen, Yuqing Zhu, Yuxi Li, Ling Luo, Qihao Zheng, Jialiang Zu, Yongchao Wu, Mindong Liu, et al. Seekbrain: An autonomous multi-agent system for accelerating neuroscience discovery. *arXiv preprint arXiv:2607.29347*, 2026.

Zhiheng Xi, Dingwen Yang, Jiaqi Liu, Jixuan Huang, Honglin Guo, Baodai Huang, Tinggang Chen, Qi Zhang, Zhonghang Lu, Chenyu Liu, et al. Agentgym2: Benchmarking large language model agents in de-idealized real-world environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44451–44479. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.2058.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38, 2026a.

Wanghan Xu, Yuhao Zhou, Yifan Zhou, Qinglong Cao, Shuo Li, Jia Bu, Bo Liu, Yixin Chen, Xuming He, Xiangyu Zhao, et al. Probing scientific general intelligence of llms with scientist-aligned workflows. *arXiv preprint arXiv:2512.16969*, 2025.

Wanghan Xu, Shuo Li, Tianlin Ye, Qinglong Cao, Yixin Chen, Hengjian Gao, Yiheng Wang, Qi Li, Kun Li, Sheng Xu, et al. Researchclawbench: A benchmark for end-to-end autonomous scientific research. *arXiv preprint arXiv:2606.07591*, 2026b.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38, 2026.

Di Zhang, Wei Liu, Qian Tan, Jingdan Chen, Hang Yan, Yuliang Yan, Jiatong Li, Weiran Huang, Xiangyu Yue, Wanli Ouyang, et al. Chemllm: A chemical large language model. *arXiv preprint arXiv:2402.06852*, 2024.

Zhongyue Zhang, Zijie Qiu, Yingcheng Wu, Shuya Li, Dingyan Wang, Yong Liu, Zhuomin Zhou, Yusong Hu, Yuhan Chen, Duo An, et al. Origene: a self-evolving virtual disease biologist automating therapeutic target discovery. *BioRxiv*, pp. 2025–06, 2025.

Xiangyu Zhao, Wanghan Xu, Bo Liu, Yuhao Zhou, Fenghua Ling, Ben Fei, Xiaoyu Yue, Lei Bai, Wenlong Zhang, and Xiao-Ming Wu. Msearth: A multimodal benchmark for earth science phenomenon discovery with mllms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5270–5301, 2026.

Lexin Zhou, Lorenzo Pacchiardi, Fernando Martinez-Plumed, Katherine M. Collins, Yael Moros-Daval, Seraphina Zhang, Qinlin Zhao, Yitian Huang, Luning Sun, Jonathan E. Prunty, Zongqian Li, Pablo Sanchez-Garcia, Kexin Jiang-Chen, Pablo A. M. Casares, Jiyun Zu, John Burden, Behzad Mehrbakhsh, David Stillwell, Manuel Cebrian, Jindong Wang, Peter Henderson, Sherry Tongshuang Wu, Patrick C. Kyllonen, Lucy Cheke, Xing Xie, and Jose Hernandez-Orallo. General scales unlock ai evaluation with explanatory and predictive power. *Nature*, 652(8108): 58–67, 2026a. doi: 10.1038/s41586-026-10303-2.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, et al. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*, 2025.

Yuhao Zhou, Yiheng Wang, Xuming He, Ruoyao Xiao, Zhiwei Li, Qiantai Feng, Zijie Guo, Yuejin Yang, Hao Wu, Wenxuan Huang, et al. Scientists' first exam: Probing cognitive abilities of mllm via perception, understanding, and reasoning. *Advances in Neural Information Processing Systems*, 38, 2026b.

Minhui Zhu, Minyang Tian, Xiaocheng Yang, Tianci Zhou, Lifan Yuan, Penghao Zhu, Eli Chertkov, Shengyan Liu, Yufeng Du, Ziming Ji, et al. Probing the critical point (critpt) of ai reasoning: a frontier physics research benchmark. *arXiv preprint arXiv:2509.26574*, 2025.

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. *Advances in Neural Information Processing Systems*, 38, 2026.

A Appendix

Appendix Contents

A.1 Agents-Eval User Guide 14
A.2 Failure Controls 16
A.3 Detailed Experimental Settings 16
A.4 Detailed Evaluation Results 16
A.5 Direct Model Evaluation 17
A.6 Capability Translation Prompts 18

A.1 AGENTS-EVAL USER GUIDE

Figure 8 illustrates the web interface for running an evaluation and inspecting its results. The steps below assume that the platform has been deployed with the required benchmark assets and execution environments.

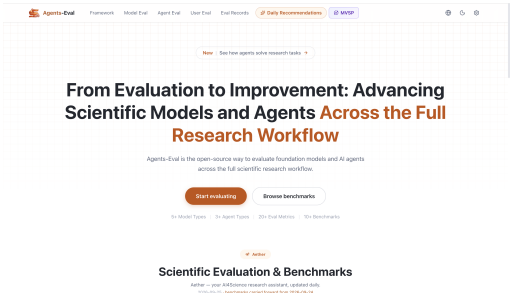
Step 1: Open the platform. On the homepage (Figure 8(a)), choose *Start evaluating* to enter the evaluation workflow or *Browse benchmarks* to inspect the available benchmarks.

Step 2: Configure services. Open *Settings* (Figure 8(b)) and add the target model under *Target Model Registry*. Specify its model identifier and API endpoint or local model path, together with any required credentials. Configure result storage and web-tool services on the same page as needed.

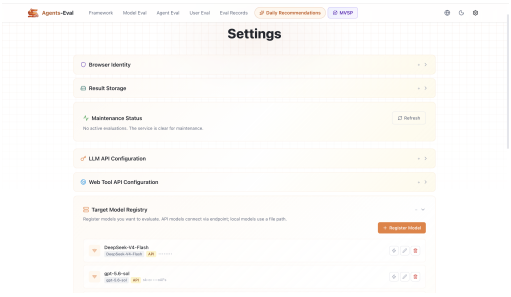
Step 3: Select and launch an evaluation. Open *Agent Eval* (Figure 8(c)) and select a benchmark. In its configuration view, choose a registered model and a compatible harness, adjust the task subset and execution parameters where supported, and submit the run.

Step 4: Inspect evaluation records. In *Eval Records* (Figure 8(d)), select a run from the sidebar to inspect its aggregate score, duration, task status, and individual task scores. Use the available logs and error information to investigate failed tasks.

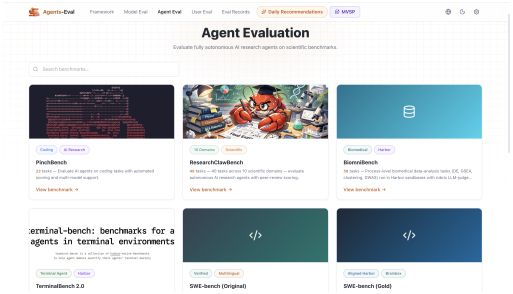
Step 5: Analyze capability demand levels. For a completed run, choose *Generate Translation Matrix* in the records view to obtain capability-based analysis. The report in Figure 8(e) shows how task success varies with capability demand level, including fitted curves and uncertainty. These curves describe whole-task outcomes grouped by capability demand. Selected runs also support diagnostic, model-comparison, and comprehensive reports that can be exported as Markdown.



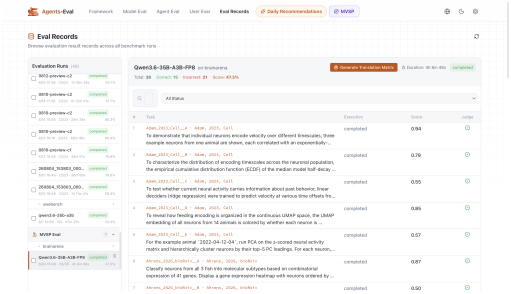
(a) Homepage



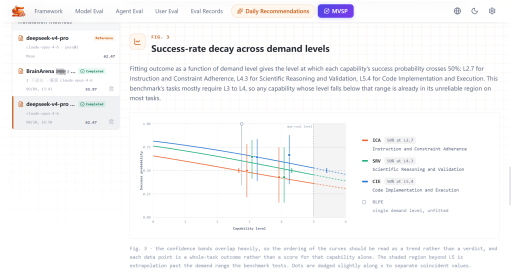
(b) Service configuration



(c) Agent benchmark selection



(d) Evaluation records



(e) Capability analysis report

Figure 8: Agents-Eval user interface, from service configuration and benchmark selection to evaluation records and capability analysis. Panels (a)–(e) correspond to the five steps in the user guide.

A.2 Failure Controls

Table 2: Evaluation problems handled across the Agents-Eval evaluation lifecycle and the mitigation applied to each.

Evaluation problem	**General mitigation**
Model response or agent execution failure	Validate and sanitize model responses; repair malformed tool calls; restart recoverable agent processes; preserve failure evidence
Model or agent timeout	Apply bounded retry and checkpoint-based resume
Model service or network request failure	Classify transient and permanent errors; retry connection failures, rate limits, and server errors with backoff; fail fast on authentication, authorization, and malformed requests
Sandbox startup, capacity, or disappearance	Run environment preflight checks; wait for quota and capacity; recreate unavailable sandboxes; remove orphaned instances
Workspace, test-data, or file-transfer failure	Use pinned images and manifest validation; preinstall runtimes and dependencies; verify files with checksums and reliable transfer procedures
Verifier dependency, execution, or artifact failure	Validate verifier dependencies; distinguish dependency, runtime, timeout, and artifact errors; preserve structured exception evidence; retry only transient cases
Abnormal task termination or cancellation	Handle termination signals gracefully; stop process groups; clean up allocated resources; resume incomplete tasks from checkpoints
Context overflow or compaction failure	Apply budget-aware compaction; retry summarization; filter oversized messages; use a deterministic fallback when compaction fails

A.3 Detailed Experimental Settings

We specify the task sets, harnesses, and execution budgets for each benchmark below.

SWE-bench Verified. We evaluate agents on the 500-task Verified split using Claude Code, mini-SWE-agent, and OpenHands. For Qwen with Claude Code, we execute each task once. We set the agent and environment setup timeout multipliers to 3.6 and 3.4, respectively. For the other runs, we use the same timeout multipliers and allow up to eight concurrent trials. We execute mini-SWE-agent and OpenHands in sandbox environments.

Terminal-Bench. We evaluate Qwen with Terminus-2 on 89 tasks. We set temperature to 1.0, top-p to 0.95, top-k to 40, presence penalty to 1.1, and reasoning effort to medium. We set `max_turns=100`, a one-hour execution timeout per task, and a concurrency limit of ten.

ALE-CLI. We aggregate all completed and timed-out tasks for each model–harness configuration. Score is the mean task score and pass rate is the fraction of tasks receiving full credit, both computed over this same set. Other execution failures are excluded. For Qwen with Claude Code, we configure a context window of 262,144 tokens, a maximum output length of 65,536 tokens, and reasoning effort of `xhigh`. We allocate four CPUs and 16 GiB of memory to each Brainbox sandbox and impose an eight-hour wall-clock limit per task.

ResearchClawBench. We evaluate Qwen3.8-27B and DeepSeek-V4.1-Flash with Claude Code on 40 tasks spanning ten domains, with a three-hour timeout per task. We report the mean following the official metrics. Timed-out runs and other execution failures without valid scores are excluded from this average.

A.4 Detailed Evaluation Results

Table 3 summarizes the 29 evaluated benchmark–model–harness combinations. ALE-CLI reports both pass rate and score, while the other benchmarks report their benchmark scores. ALE-CLI uses

the completed and timed-out tasks described above. For SWE-bench Verified and Terminal-Bench 2.1, tasks without a valid result are counted as unsuccessful. ResearchClawBench reports the mean over valid evaluations, as specified above. Dashes indicate combinations that were not evaluated.

Table 3: Raw experimental results reported in this paper.

Benchmark	**Harness**	**Qwen-3.8-27B**		**DeepSeek V4.1-Flash**		**GLM-5.3**		**GPT-6-luna**	
		Pass rate	Score	Pass rate	Score	Pass rate	Score	Pass rate	Score
ALE-CLI	Claude Code	16.50	39.09	24.51	49.17	–	–	26.60	53.74
	ALE-Claw	20.39	38.92	27.18	49.60	–	–	10.00	34.58
	Codex	19.23	47.75	35.24	59.34	–	–	19.42	46.51
		Score		Score		Score		Score	
SWE-bench Verified	Claude Code	94.20		93.60		91.20		–	
	OpenHands	72.20		86.00		91.40		–	
	mini-SWE-agent	84.80		89.60		86.40		–	
Terminal-Bench 2.1	Claude Code	70.79		80.90		86.52		–	
	Terminus-2	68.54		84.27		87.64		–	
	mini-SWE-agent	71.91		80.90		83.15		–	
ResearchClawBench	Claude Code	25.46		20.65		–		–	

Score < 0.8 Score ≥ 0.8

Task ID	Task score Run 1	Run 2	Run 3	High-score runs	Capability demand level ICA	TC	EISS	RLPE	SRV	CIE
business_finance/financial_stmt_reconstruction_aapl_fy2024	0.000	1.000	1.000	2 of 3	2	2	0	0	0	0
computing_math/branch_bound_atsp	1.000	1.000	0.000	2 of 3	2	3	0	2	0	4
computing_math/ranking_node_feature_parity_recovery_instance_1	1.000	1.000	0.000	2 of 3	3	4	0	2	0	3
computing_math/recsys_cold_start_instance_1	0.833	0.500	1.000	2 of 3	3	3	0	3	0	4
education_info/moodle_gradebook_closeout_reconciliation	0.000	0.950	0.950	2 of 3	3	4	0	2	0	3
health_medicine/epidemiology_forecast	1.000	1.000	0.000	2 of 3	3	3	0	0	1	3
health_medicine/healthcare_variant_annotation_pipeline	0.999	0.794	0.999	2 of 3	2	3	0	1	0	3
health_medicine/replicate_paper_1	0.000	1.000	1.000	2 of 3	3	4	0	2	3	3
life_sciences/WGS_Variant_Calling	0.900	0.900	0.300	2 of 3	2	4	0	2	3	2
life_sciences/hg002_chr22_germline_variant_pipeline	0.950	0.900	0.000	2 of 3	3	4	0	2	3	3
life_sciences/protein_function_annotation_instance_1	1.000	0.000	1.000	2 of 3	0	4	4	0	0	0
life_sciences/tms_marrow_cell_type_annotation_instance_1	1.000	0.000	1.000	2 of 3	2	3	0	2	4	3
physical_sciences/adapt_vqe_molecular_energy	0.000	1.000	1.000	2 of 3	2	4	0	2	4	4
business_finance/bpmn_supply_disruption_l3	0.000	0.904	0.000	1 of 3	4	5	0	3	0	4
business_finance/internal_employee_agent_instance_1	0.000	0.000	1.000	1 of 3	0	0	0	0	0	0
computing_math/cfr_game_theory_equilibrium	0.670	1.000	0.670	1 of 3	4	4	0	2	0	5
computing_math/paper_reproduction_instance_1	0.785	0.785	0.800	1 of 3	2	5	2	2	0	3

Figure 9: ALE-CLI tasks with high scores in one or two of three Qwen3.8-27B runs, supplementing Figure 6. Each row shows task scores and capability demand levels; blue indicates a score of at least 0.8 and gray a lower score.

A.5 Direct Model Evaluation

Agents-Eval also evaluates language and multimodal models through direct endpoint queries. Table 4 lists the benchmarks marked as verified in the model-evaluation catalog. Each benchmark retains its own evaluation procedure, including code execution or model-based judging where required. These procedures evaluate the model's responses without requiring an agent harness.

Table 4: Verified benchmarks for model evaluation in Agents-Eval.

Task domain	Benchmark
Instruction Following	IFBench Pyatkin et al. (2026)
Coding	SciCode Tian et al. (2024)
Scientific Reasoning	ChemBench Zhang et al. (2024)
	CMPhysBench Wang et al. (2026b)
	TRQA Zhang et al. (2025)
	CritPt Zhu et al. (2025)
Scientific Multimodal Understanding	MSEarthMCQ Zhao et al. (2026)
	SFE Zhou et al. (2026b)
	AstroVisBench Joseph et al. (2026)

A.6 Capability Translation Prompts

The following six boxes reproduce the complete dimension-specific rubrics used by the capability translation pipeline. Each annotation call receives one task description and one rubric and assigns a capability demand level from 0 to 5. The judge assesses task requirements without observing agent responses, execution trajectories, or evaluation outcomes. Each rubric includes a dimension definition, six demand-level descriptions, and calibration examples. Within the reproduced rubrics, "level" denotes the capability demand level, not an agent proficiency level.

Human review and calibration. We assessed the rubrics through iterative human review of benchmark examples and pilot annotations. The reviewer examined whether assigned demand levels reflected the task requirements, identified ambiguous demand-level boundaries and conflation between dimensions, and guided revisions to the definitions and examples. This process involved more than 20 rubric revision rounds and more than 10 small-batch pilot annotation rounds, with targeted re-annotation of affected dimensions during calibration. Human feedback helped distinguish instruction constraints from implementation complexity, tool orchestration from coding difficulty, and individual research operations from complete research lifecycles. This provides qualitative evidence of the rubrics' interpretability and alignment with task demands, rather than an independent estimate of annotation accuracy.

ICA: Instruction and Constraint Adherence

This criterion assesses the demand a task places on an agent to identify, retain, coordinate, and satisfy explicit constraints on how the requested result must be expressed, structured, ordered, bounded, or maintained. These constraints may govern required or forbidden content, counts, output format, position, language or style, conditional scope, or rules active across parts or turns. The level is judged from the number and precision of distinct constraints, their persistence and interaction, and the difficulty of satisfying them together. Technical steps, filenames, paths, or schema fields that independently describe one coherent implementation contract should not each be counted as separate instructions. Constraints do create additional demand when values, identifiers, parameters, commands, or references must remain consistent across several artefacts or scopes. A long task description, many deliverables, or difficult underlying work does not by itself create high ICA demand.

Level 0: None. The task states a core goal but imposes no additional requirement on how the result must be expressed, structured, ordered, bounded, or maintained. Many reasonable output forms would satisfy the instruction. *Examples:*

- "Explain why leaves change colour in autumn."
- "Summarise the following article."
- "Find the cause of the error in this program."

Level 1: Very low. The task contains one simple, direct, and static requirement in addition to its core goal. The requirement may be local or apply to the whole output, but it can be checked independently and does not need to be coordinated with another rule. A larger count or length threshold remains at this level when it is still only one straightforward requirement. *Examples:*

ICA: Instruction and Constraint Adherence (continued)

- "Write a resume for a recent high-school graduate and include at least twelve placeholders enclosed in square brackets."
- "Explain what Lua code is and use at least eighty-five distinct words in the response."
- "Write a withdrawal application of at least 300 words."

Level 2: Low. The task contains a small set of compatible requirements that can be planned as one stable output or delivery template, or a shallow conditional rule with clear local branches. The requirements may concern several aspects of the result, but they remain aligned and easy to check in one pass. A coherent engineering contract containing several compatible filenames, locations, fields, or validation checks normally remains at this level when each requirement can be verified locally using one short checklist. *Examples:*

- "Summarise the news in bullet points and use no more than 100 words."
- "List the TCP/IP layers as a two-level bullet list, give one commonly used protocol for each layer, and describe each protocol in no more than fifteen words."
- "For each country, if it has Special Drawing Rights, write a withdrawal application of at least 300 words in bullet points; otherwise explain the reason in no more than 100 words."

Level 3: Intermediate. The task requires moderate coordination because instructions have an exact order or position, govern different parts of the output, remain active across conversation turns, or distribute exact requirements across several artefacts whose shared values and references must agree. Individual rules may still be simple, but they cannot be satisfied reliably without retaining their scope and coordinating when and where they apply. *Examples:*

- "First repeat the specified request word for word, with no character before the repetition, and then write the requested email with a title enclosed in double angle brackets."
- "In the first turn, answer only in lowercase English; in the second turn, add a title enclosed in double angle brackets; in the third turn, wrap the complete answer in double quotation marks while preserving the earlier requirements."
- "Preserve the previously specified language rule, produce at least 801 words, and end with the exact phrase 'what are they doing?' with nothing after it."
- "Create three workflow files with specified names and parameters, reuse one exact basename throughout them, and ensure that every generated path and command-line input refers to the matching artefact."

Level 4: High. The task requires substantial tracking or bookkeeping across many distinct and precise constraints. It may assign different exact counts to several elements, combine persistent rules with different scopes, or require a final global check over the complete output. The principal difficulty is maintaining numerous independent targets. A lengthy engineering specification does not reach this level when its steps and artefacts form one coherent implementation contract. *Examples:*

- "Use 'kaleidoscope' once, 'nebula' twice, 'whisper' three times, 'labyrinth' five times, and 'paradox' seven times."
- "Write a resume enclosed in double quotation marks, include a title in double angle brackets, use at least one but fewer than ten all-capital words, produce exactly six paragraphs, and separate the paragraphs with three asterisks."
- "Write five labelled sections, give each section a different exact word limit and required keyword, preserve the specified section order, and verify all section-level counts in the final response."

Level 5: Very high. The task requires difficult constrained construction in which several requirements are tightly coupled. Satisfying one rule makes other rules harder to satisfy, so the response must be planned and revised globally rather than produced by applying independent checks. Typical demands include simultaneous phonetic, morphological, lexical, structural, and length constraints; interacting nested scopes; or exact local rules whose satisfaction must remain globally consistent. *Examples:*

- "List the TCP/IP layers and one real-world protocol for each in a two-level bullet structure. Describe every protocol in no more than fifteen words, with both alliteration and an assonance pattern in every description."
- "Write three haiku about gravity, each following a 5-7-5 syllable pattern, using a different scientific term and at least one word beginning with 'geo-', while also placing alliteration in at least one line of every haiku."
- "Produce several sections under a strict total word budget, satisfy different local lexical and formatting rules in each section, and maintain exact cross-section word counts and reference relationships after every necessary revision."

TC: Tool Complexity

This rubric assesses the complexity of the tool workflow required to complete a task, ranging from no tool use to extensive orchestration across heterogeneous tool ecosystems. Tools include functions and APIs, search and database interfaces, command-line programs, browsers and graphical interfaces, file-processing utilities, scientific or engineering software, and libraries invoked through code. The score is based on the simplest reasonable tool workflow that satisfies the task. It is judged holistically from the number and dependency of necessary calls, the diversity and difficulty of tools that must be operated, and the interaction and state complexity of the workflow, including result-dependent branching, persistent state, partial-failure recovery, and final consistency checks. Reading several ordinary document formats, performing a deterministic calculation, and writing a structured result remain one predictable workflow unless they require materially different specialist tools or create consequential state dependencies.

Level 0: None. The task can be completed from the supplied information through reasoning or response generation alone. It does not require calling an external function, searching a source, opening or transforming an artefact with a tool, operating an interface, or writing code that invokes a tool or specialist library. *Examples:*

- "Explain the conclusion using only the passage supplied in the prompt."
- "Rewrite this paragraph in a more formal style."
- "Calculate the result from the displayed formula without consulting or operating any external resource."

Level 1: Very low. The task requires one simple call, or a very small number of independent calls, using one familiar tool type. The correct tool is obvious, the required arguments are explicitly supplied or directly copied from the task, and the call does not require custom orchestration code, specialist configuration, substantial interpretation, or tracking of changing state between calls. *Examples:*

- "Call `calculate_triangle_area` with base 10 and height 5 and return the result."
- "Use the named search function once to look up the supplied reservation number."
- "Open the specified file with the provided reader and return the value of one named field."

Level 2: Low. The task requires several routine calls of one tool type, or a short and mostly predictable chain involving a small number of tool types. Some returned identifiers or values may be passed to later calls, and a simple lookup, extraction, calculation, comparison, state change, or transaction may be required, but the sequence and state transition are clear in advance, the tools are familiar, and little or no custom calling code is needed. Repeated homogeneous calls, routine pagination or polling, a simple search followed by one action, or sequential document reading followed by deterministic calculation and structured output normally remain at this level. *Examples:*

- "Look up the reservation and customer membership record, then use the two returned records to report the baggage allowance."
- "Search for an available product matching the stated attribute and exchange the original item for the selected variant."
- "List the files, run the same provided conversion command on each matching file, and collect the outputs."
- "Read the supplied PDF rules and JSON case records, apply the stated fee schedule to each case, and save the calculated fields in the supplied JSON schema."

Level 3: Intermediate. The task requires a moderately long but bounded sequence, several materially different tool operations, or limited adaptive interaction in which later calls, arguments, or actions depend on earlier results. The agent may need to preserve local state across a branch, verify an operation, recover from a routine failure, write a modest script to invoke APIs or read tool-specific files, or operate a documented professional tool requiring domain knowledge. Merely crossing common document formats does not reach this level; the operations themselves must require meaningful adaptation, specialist operation, or dependent intermediate state. Several tools can remain at this level when the workflow is mostly sequential, intermediate state is local, and completion follows a clear plan without substantial cross-tool recovery or reconciliation. *Examples:*

- "Find the matching historical flight, search the new date, calculate the price, remove the second passenger if the price exceeds the limit, choose the requested adjacent seats, and create the reservation."
- "Write Python that reads `vasprun.xml` and `LOCPOT`, constructs the required Pymatgen objects, calls `get_freysoldt_correction` with the specified physical parameters, and extracts the correction energy with local error handling."
- "Query a database for its schema, construct a second query from the returned fields, export the selected records with another tool, and verify the exported row count."

Level 4: High. The task requires substantial orchestration across multiple distinct or specialist tools, with dependent intermediate artefacts or interacting persistent states that must remain consistent. Correct completion depends on non-trivial custom orchestration, specialist configuration or interpretation, and validation

TC: Tool Complexity (continued)

across stages. The workflow includes meaningful result-driven iteration, non-routine failure diagnosis and recovery, or reconciliation across tools or states. The defining demand is managing interactions among tools: each stage can change what later stages must do, and a partial or inconsistent result must be repaired before the workflow can complete. *Examples:*

- "Read several VASP and PROCAR datasets, construct `HarmonicDefect` objects through two different Pymatgen workflows, compare their derived properties, deliberately test a non-unique-spin failure case, and return validated results with field-level error handling."
- "Process several defect charge states, calculate a correction for each state, build defect entries and a phase diagram, construct a formation-energy diagram, and compute formation energy and concentration under specified thermodynamic conditions."
- "Synchronize records between two stateful services, transform their schemas with a custom script, resume after a partially completed transfer, resolve rejected or conflicting records, and verify counts and identifiers in both systems."

Level 5: Very high. The task requires an extensive, sustained, and dynamic tool-driven workflow combining many calls, multiple heterogeneous or specialist tool ecosystems, difficult custom orchestration, and strongly interacting state changes. The agent must operate expert-level tools, create or repair complex intermediate artefacts, monitor asynchronous or concurrent operations, repeatedly replan from new observations, recover from consequential failures or competing updates, and perform independent end-to-end validation against an explicit convergence or consistency condition. A task should reach this level because high demand is present across tool scale, diversity, difficulty, and interaction or state complexity together, rather than because of call count, specialist terminology, or one difficult call alone. *Examples:*

- "Repair defects in a DICOM structure set, import the case into matRad, configure and run fluence optimisation and sequencing, calculate dose, export mutually consistent RTPLAN, RTDOSE, and RTSTRUCT files, then independently recompute DVH metrics in Python and verify the plan with generated reports and figures."
- "Build and execute a scientific pipeline spanning simulation software, several domain-specific file formats, custom analysis code, parameter adjustment from intermediate results, and an independent validation toolchain, monitoring long-running jobs and repeatedly recovering failed stages without corrupting later artefacts."
- "Coordinate a live production migration across several APIs, databases, command-line tools, and monitoring systems, write the required adapters, handle competing updates and partial failures, and continue reconciliation until every system satisfies the final consistency conditions."

EISS: External Information Search and Synthesis

This rubric assesses the complexity of discovering, retrieving, evaluating, connecting, and synthesising information outside the task materials and execution environment. The level represents the demand of the evidence-acquisition process, ranging from no external search to open-ended investigation across sparse, changing, or poorly indexed sources. It is judged holistically from the breadth of the search space; the depth, persistence, and reformulation required to locate evidence; the diversity and authority of the necessary sources; and the difficulty of connecting evidence, resolving conflicts, eliminating candidates, and establishing sufficient coverage. Source count alone does not determine the level.

Level 0: None. The supplied task materials and internal environment contain all information needed for a valid completion, so no external source needs to be discovered, opened, or consulted. *Examples:*

- "Compute the statistic from the table included in the prompt."
- "Use the supplied local files to produce the requested plot."
- "Answer the question using only the provided passage."

Level 1: Very low. The task requires directly retrieving one clearly specified fact or record from an explicitly named source, exact page, or known location. The source-selection decision is effectively given, navigation is minimal, and no search strategy, candidate comparison, or cross-source synthesis is needed. *Examples:*

- "Open the named official release page and report the listed publication date."
- "Retrieve the stated indicator for the supplied country and year from the named database table."
- "Use the provided DOI to open the publisher record and return the article title."

Level 2: Low. The task requires a focused search within a constrained and well-defined information space for one or a few facts. The query conditions are specific, the likely sources are easy to identify, and the answer can be obtained through a short search or shallow navigation path, possibly followed by a simple combination of one or two compatible sources. Little query reformulation, source evaluation, or candidate elimination is required. *Examples:*

- "Find which Dutch player scored from open play in the specified 2022 World Cup match."

EISS: External Information Search and Synthesis (continued)

- "From the named conference website, find the Industry Track submission deadline and the venue address on two official pages."
- "Find the official documentation for two named functions and combine their stated parameter requirements."

Level 3: Intermediate. The task requires several dependent searches, deeper navigation within a bounded source space, or evidence distributed across several sources. The agent must reformulate at least some queries or follow intermediate findings, retain and connect multiple facts, and check that the gathered evidence jointly answers the question. The target and stopping condition remain clear, and extensive source-quality adjudication or open-ended candidate discovery is not required. *Examples:*

- "Navigate the conference website to collect the daily schedules for an activity across six dates, align the time intervals, and provide the evidence needed to calculate the total attendance time."
- "Find the year of a specified Best Actor award, move sixteen years back, and identify the Best Costume Design winner for that year from authoritative award records."
- "Trace a paper to its cited dataset and method documentation, reconcile the relevant definitions, and report the linked facts with their sources."

Level 4: High. The task requires a difficult but bounded investigation involving many searches, deep traversal, numerous candidates, heterogeneous or specialised sources, or substantive source-quality and compatibility checks. The agent must discover and eliminate plausible alternatives, connect a long chain of clues, resolve conflicting or mismatched evidence, or build a systematic synthesis whose target class and practical search boundary are nevertheless identifiable in advance. *Examples:*

- "Starting from an official university website, traverse multiple branches with a combined depth of six to eight pages to identify a project lead, the latest funding amount, and the lead's laboratory address, then verify that all three records refer to the same project."
- "Identify an obscure work from several indirect biographical, publication, and archival clues when no page states the complete answer, testing and eliminating many near-matching candidates."
- "Retrieve evidence from numerous specialised articles and databases, harmonise incompatible definitions, explain substantive conflicts, and produce a source-traceable comparison within a specified topic and time period."

Level 5: Very high. The task requires an open-ended, iterative, and search-intensive investigation in which the relevant candidates, sources, or stopping point are not known at the outset. Evidence may be sparse, changing, fragmented, duplicated, or poorly indexed. The agent must repeatedly revise the search strategy, expand or narrow the candidate space, assess authority and independence, manage unresolved uncertainty and coverage gaps, and produce a defensible synthesis or evidence map with explicit limits. Tasks at this level usually combine extreme search difficulty with broad evidence integration; one obscure lookup alone is insufficient. *Examples:*

- "Systematically discover the leading teams at the intersection of mathematics and quantum computing, compare their research, publications, international collaboration, funding, and industry partnerships, then assess which teams may drive breakthroughs over the next decade and support the forecast with traceable evidence."
- "Investigate a rare historical entity through fragmented archives and indirect relational clues, repeatedly changing search paths as candidates fail, distinguishing independent evidence from copied claims, and documenting why the final answer is the only defensible match."
- "Maintain a time-stamped evidence synthesis for an emerging event with incomplete and contradictory reporting, detect derivative claims, revise confidence and coverage as new sources appear, and state what unresolved evidence could change the conclusion."

RLPE: Research Lifecycle Planning and Execution

This criterion assesses the demand of planning and carrying out a scientific or scholarly research lifecycle, or substantive stages within one. Research addresses a study question, hypothesis, or knowledge gap through a methodological process: literature review, research or experimental design, systematic observation or data collection, experimental or theoretical investigation, result analysis, validation, and interpretation. Each stage has a methodological purpose and a research output that contributes to the study. The level reflects the completeness and integration of these stages, the complexity of the research methods, and the autonomy required to carry them out.

Level 0: None. The task requires no research-stage operation or research-method decision. *Examples:*

- "Find a person's university graduation year using the available reference sources."
- "Identify the object that matches the supplied factual descriptions."
- "Convert the supplied table into the requested JSON schema."

RLPE: Research Lifecycle Planning and Execution (continued)

- "Implement the specified function and verify it against the provided tests."

Level 1: Very low. The task requires one bounded operation within a defined scientific or scholarly study. The operation contributes directly to the study's evidence, measurements, literature review, experiment, or analysis, under a supplied research objective and procedure. Its output is a research input or result, such as a coded observation, an extracted study outcome, an experimental measurement, or a descriptive result from study data. *Examples:*

- "For the stated experiment, calculate the mean of the supplied measurements and report the value."
- "Track the specified biological structures across the supplied image frames to create the trajectory data required by the study."
- "For the stated literature-review question, extract the sample size and principal finding from the supplied study."
- "Apply the supplied coding scheme to interview excerpts for a study of changes in workplace practices."

Level 2: Low. The task requires two or three distinct adjacent stages of a defined scientific or scholarly study, connected by their research outputs. For example, a specified experiment or simulation produces data for a separate analysis, or a systematic literature review produces a body of study results for methodological synthesis. The research objective and main method are supplied, and the work follows a largely prescribed path. Stage boundaries follow changes in research purpose and method, with each stage contributing a distinct study output. *Examples:*

- "For the stated experimental objective, fit the prescribed model to the supplied measurements, assess its fit and uncertainty, and interpret the result for the study question."
- "Reconstruct the supplied physical scenario with the prescribed model, simulate the measured response, check sensitivity to the specified numerical setting, and interpret the bounded result."
- "For the specified systematic-review question, select studies using the supplied eligibility criteria, extract their effect estimates, and synthesise the results with the prescribed method."

Level 3: Intermediate. The task requires a bounded and substantially complete research process with a defined research question or objective. It includes reviewing relevant evidence, selecting or justifying a method, conducting an experiment or investigation, analysing and validating the results, and forming a supported conclusion. The research scope is clear and the main methodological choices are constrained. *Examples:*

- "For the stated hypothesis, review the supplied background studies, define measurable variables, choose an appropriate analysis, evaluate model diagnostics, perform a robustness check, and report the supported interpretation."
- "Using the provided dataset and research objective, design and conduct a controlled analysis, test a competing explanation, validate the result on a held-out sample, and state the conclusion."
- "Plan a bounded laboratory study for the specified question, select measurements from the available instruments, execute the experiment, quantify uncertainty, and interpret the findings."

Level 4: High. The task requires a complete research lifecycle for a broad research problem. The agent must independently conduct literature review, formulate hypotheses or precise objectives, design the research or experiments, select or collect evidence, execute the investigation, analyse and validate the results, and synthesise a defensible conclusion. Several substantive research decisions and technically demanding methods are required. *Examples:*

- "Review black-hole superradiance research, formulate a probabilistic constraint model that uses full mass-and-spin posterior distributions, apply it across black holes at different mass scales, test sensitivity to assumptions, and interpret the resulting constraints on ultralight-boson properties."
- "Investigate systematic differences among glacier-observation methods, design an uncertainty-aware reconciliation procedure, combine hundreds of regional estimates into global time series, validate consistency across methods, and interpret the implications for climate assessment."
- "Develop a global risk study that combines changes in tropical-cyclone regimes with sea-level rise, integrates mangrove and climate-scenario datasets, validates a composite risk index, maps future exposure, and synthesises implications for conservation decisions."

Level 5: Very high. The task requires leading an open-ended and highly complex research programme through the complete research lifecycle. The agent must discover or substantially refine the research problem and autonomously coordinate extensive literature review, multiple hypotheses, advanced or novel methodological and experimental design, substantial evidence or data collection, several interdependent experiments or investigations, rigorous result analysis and validation, and an integrated account of findings, limitations, and unresolved questions. *Examples:*

- "Lead an inverse-design programme for recyclable vitrimeric polymers by combining molecular-dynamics simulations, Gaussian-process calibration, and a graph generative model, exploring new

RLPE: Research Lifecycle Planning and Execution (continued)

chemistries, selecting candidates with target glass-transition temperatures, experimentally validating them, and using the results to refine the research models."
- "Build an autonomous materials-discovery programme that pretrains representations on a large unlabeled crystal collection, learns from scarce positive examples, screens unknown structures, validates candidates with first-principles calculations, and characterises the electronic and magnetic properties of newly discovered materials."
- "Establish and test a general theory for designing multi-component icosahedral nanoclusters by exploring element combinations, shell paths, and size mismatches, validating stability with several interaction models and first-principles calculations, running self-assembly growth simulations, and synthesising design rules across material families."

SRV: Scientific Reasoning and Validation

This rubric assesses the depth and difficulty of reasoning required to answer a substantive scientific question in the natural sciences, life sciences, or engineering sciences. The level is judged from the depth of scientific knowledge required; the number and interaction of relevant concepts, mechanisms, variables, and assumptions; the difficulty of interpreting measurements and evidence; and the reasoning needed to compare explanations, handle uncertainty, or validate a scientific conclusion. Mathematics, statistics, optimisation, simulation, or experimentation used only for finance, business, software, or operations does not create SRV demand. Engineering science concerns scientific questions about physical or technical systems, not software implementation by itself.

Level 0: None. The task does not ask a substantive scientific question. A non-scientific task remains at this level even if it involves difficult mathematics, statistics, optimisation, simulation, experimentation, search, coding, or a long workflow. *Examples:*

- "Implement a Monte Carlo model for pricing an American financial option and validate its numerical output."
- "Analyse a marketing A/B test with significance tests and recommend whether to ship the campaign."
- "Redesign and test a business-process workflow in a software execution engine."

Level 1: Very low. The task asks a basic scientific question that can be answered with elementary scientific literacy, direct factual knowledge, an explicit observation, or one simple relationship. It requires little or no multi-step scientific reasoning, and the relevant concept or comparison is apparent from the task. *Examples:*

- "State whether the measured temperature increased or decreased in the supplied graph."
- "Identify which sample has the higher concentration from the explicitly labelled measurements."
- "Using the two values stated in the table, calculate the ratio of upregulated crotonylated sites to upregulated succinylated sites."

Level 2: Low. The task requires routine application of established scientific knowledge at an introductory university or ordinary technical level. One or a few standard principles, formulas, or interpretations must be selected and applied through a short, well-defined reasoning path, with limited interaction among assumptions or sources of evidence. *Examples:*

- "Calculate the entropy change when a stated quantity of heat is transferred reversibly and isothermally at a given temperature."
- "Use the ideal-gas law to calculate the missing state variable and check that the units and magnitude are plausible."
- "Use a standard dose-response relationship to determine which supplied concentration reaches the stated effect threshold."

Level 3: Intermediate. The task requires advanced university, graduate-level, or routine professional scientific knowledge and a substantive but bounded chain of reasoning. Several scientific concepts, variables, assumptions, or analysis steps must be connected, but the relevant framework is established, the evidence is reasonably well behaved, and an appropriately trained practitioner can follow a clear solution path without resolving many tightly coupled ambiguities. *Examples:*

- "Derive the internal-energy change for a real-gas expansion from the supplied equation of state, applying the correct thermodynamic identity and boundary conditions."
- "Compare two established kinetic models against the supplied measurements, check their assumptions, and determine which model is better supported."
- "Interpret a set of mutually consistent spectroscopy and microscopy results to identify the most likely material phase and verify it with a standard diagnostic check."

Level 4: High. The task requires difficult expert scientific reasoning over multiple interdependent mechanisms, variables, assumptions, measurements, or procedural choices. Correct completion depends on specialised domain knowledge, careful tracking of interacting constraints, diagnosis of a scientifically meaning-

SRV: Scientific Reasoning and Validation (continued)

ful failure or ambiguity, or integration of several forms of evidence. The problem may have an established answer, but reaching it requires more than routine application of a known method. *Examples:*

- "After an iPSC lipid-mediated transfection protocol produces low efficiency, inspect the protocol, identify the scientifically consequential step that caused the result, and explain why the proposed correction should work."
- "Given a long DNA sequence and multiple restriction enzymes, track every recognition site, strand orientation, and circular or linear boundary to determine all resulting fragment lengths."
- "Use several direction-dependent spectroscopic and thermodynamic observations to determine which candidate superconducting order parameter is compatible with the joint evidence and rule out the alternatives."

Level 5: Very high. The task is an exceptionally demanding scientific problem requiring sustained expert reasoning across many tightly coupled scientific elements. It may involve long dependent chains of mechanistic inference, numerous interacting sequences or variables, several plausible explanations or failure modes, difficult measurement or experimental constraints, and rigorous cross-checking of the final conclusion. Small errors in an early scientific judgement materially change later conclusions, and even a domain specialist may need substantial time and careful external calculation or analysis to solve the problem reliably. *Examples:*

- "Given four complete plasmid sequences, determine the product of a multi-part Golden Gate assembly, propagate all junction and orientation constraints, simulate diagnostic digestion with two enzymes, and identify the fragment pattern that proves the clone is correct."
- "Reconcile several high-precision experimental measurements affected by different systematic biases, trace how the competing mechanisms predict each observation, and determine which explanation remains consistent after all uncertainty and robustness checks."
- "Diagnose a failed multi-stage scientific experiment in which several coupled preparation, measurement, and calibration choices can produce similar symptoms, determine the jointly consistent failure pathway, and specify independent checks that verify the diagnosis."

CIE: Code Implementation and Execution

This rubric assesses the difficulty of the code, executable configuration, or computational implementation that the agent must create or modify to complete a task. The score is based on the simplest reasonable implementation path that satisfies the task requirements. If no executable implementation is needed, the task is Level 0. If code is involved only as a small auxiliary script, command, configuration change, or local edit, the task is Level 1. Levels 2 through 5 require substantive implementation as a necessary part of the solution. Difficulty is determined by the logic and structure that must be implemented, integration across functions, files, data representations, or components, the diagnosis and debugging required, and the testing, reliability, performance, and reproducibility of the resulting implementation. Configuring and iteratively running established software remains at Levels 1 or 2 when the agent need not implement substantial analysis, orchestration, or algorithmic logic. Multiple output files, plots, or reproducibility requirements do not by themselves raise an implementation to Level 4. Tool use and domain reasoning affect CIE only to the extent that they create implementation requirements.

Level 0: None. Correct completion needs no code, executable configuration, or computational implementation. The required result can be produced directly through reasoning, retrieval, selection, or content construction. *Examples:*

- "Explain the supplied concept in two paragraphs."
- "Choose the correct option from the list and justify the choice."
- "Summarise the supplied report without writing or running code."

Level 1: Very low. Code is limited to a small auxiliary script, simple self-contained function, expression, command, configuration change, or local code edit. The logic is elementary or directly specified, the implementation has few meaningful branches, and correctness can be checked with a small number of obvious examples. Basic automation, syntax, standard-library use, or elementary arithmetic and list or string operations remain at this level. *Examples:*

- "Write a Python function that converts a supplied string to uppercase and passes three direct tests."
- "Write a short script that reads the supplied table, selects the specified columns, and saves the result."
- "In an existing repository, change an explicitly identified constant or condition and run the one directly relevant test."

Level 2: Low. Substantive implementation is required, but it is limited to a bounded function, short self-contained module, localised repair, or straightforward application of established libraries or software through supplied interfaces. This includes non-trivial executable configuration and iterative parameter adjustment

CIE: Code Implementation and Execution (continued)

around an existing scientific or engineering program when no custom analysis system or algorithm must be implemented. The logic is non-trivial but standard, the input and expected output are clear, and any glue code is small and direct. Testing is local and limited; substantial diagnosis, custom orchestration, or cross-component reliability is not required. *Examples:*

- "Compute the median of two sorted arrays, handling the required index and boundary cases."
- "Implement a dynamic-programming function that returns the size of the largest divisible subset and verify it on supplied cases."
- "Translate the supplied equations for the Haldane model into a function that returns the specified 2-by-2 complex Hamiltonian and check several numerical inputs."
- "Tune the executable configuration of an existing simulation program against supplied observations and save the required output, without implementing the simulator or a new calibration algorithm."

Level 3: Intermediate. The task requires a substantive but bounded implementation or repair involving non-trivial diagnosis, several dependent transformations or functions, custom orchestration across multiple steps, formats, libraries, or executable configurations, meaningful integration with existing code, or explicit regression and edge-case testing. Established tools may perform the core domain computation, while the agent implements the surrounding analysis workflow and consistency logic. A reproducible statistical workflow using established methods normally remains at this level when it requires substantive code but no important custom algorithm or system component. An experienced developer can identify a coherent implementation path without constructing a large system or managing many strongly coupled failure modes. *Examples:*

- "Locate and repair the SymPy `kernS` uninitialised-variable bug, add a regression test for the reported expression, and confirm that related behaviour still passes."
- "Implement a PCA function that validates its arguments, transforms a DataFrame, returns correctly labelled results, produces the required plot, and passes interface and edge-case tests."
- "Complete a partially implemented package across several functions, preserve its public API, and add regression tests covering the original failure and relevant boundary cases."
- "Implement a reproducible statistical-analysis script that computes a specified study design with established methods and generates mutually consistent tables, machine-readable results, and plots."

Level 4: High. The task requires a difficult engineering implementation in which the agent must create an important custom algorithm, system component, or complex pipeline logic. Several stages, components, or data representations are strongly coupled, and correct completion depends on substantial debugging and integration, robust handling of irregular inputs or external failures, careful validation of intermediate and final artefacts, or material performance and reproducibility constraints. Established tools may be used as building blocks, but invoking them, configuring them, or using standard methods to generate several related artefacts is not the core implementation. Novelty is not required. *Examples:*

- "Build a self-contained program that preprocesses Clintox data, trains and evaluates a multitask molecular classifier, produces calibrated test probabilities, and saves a strictly formatted result with performance checks."
- "Train a VAE on single-cell data, perform one-vs-all differential-expression analysis, extract markers, construct a dendrogram, and generate a validated dot plot in a reproducible pipeline."
- "Implement an FTP download workflow that manages connection and authentication state, remote and local paths, subprocess downloads, cleanup, distinct failure modes, and mock-based tests."

Level 5: Very high. Engineering implementation is the dominant challenge and requires sustained work across many tightly coupled algorithms, modules, execution stages, or system states. The task demands difficult architectural decisions or technically demanding numerical or systems code, extensive diagnosis and recovery, and broad end-to-end validation under interacting correctness, scale, performance, reliability, or reproducibility constraints. A task can reach this level by implementing and validating an exceptionally complex established method; it need not invent a new algorithm or system. *Examples:*

- "Implement the complete Haldane-model Chern-number phase-diagram calculation, including Hamiltonians, eigensystems, gauge-consistent link variables, periodic grid boundaries, a two-parameter sweep, numerical stability, performance control, and domain-specific invariant checks."
- "Implement and validate a distributed execution service with scheduling, checkpointing, concurrency control, recovery from partial failure, observability, and end-to-end consistency tests."
- "Reconstruct a missing computational stack from incomplete components, integrate several specialised representations and execution stages, support restart and recovery, and demonstrate reproducibility across environments and workloads."