CSFORGE: AUTONOMOUS CONSTRUCTION OF VERIFIABLE ENVIRONMENTS FOR COMPUTER SCIENCE RESEARCH AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents are increasingly being developed for end-to-end computer science research (CSR), with further progress depending on reliable environments that can be constructed at scale to support their evaluation and improvement. Existing studies provide essential resources for evaluating research agents, but converting these resources into executable tasks remains challenging, as research objectives, experimental constraints, and evaluation protocols should be made explicit. We introduce CSForge, an automated pipeline that transforms published research papers and their released artifacts into verifiable CSR environments through three stages: paper filtering, task synthesis, and environment deployment. Drawing on human research experience, CSForge formulates structured research tasks and refines their evaluation protocols through iterative attack-and-repair to address shortcut solutions. It then reproduces reference experiments and constructs executable environments, checking their consistency with the source artifacts, reported findings, and repeated runs. Across sampled filter-passing papers from seven CS subfields, 86.8% of construction attempts yield executable environments, and 80.0% yield environments that pass human quality review. Experiments with six frontier models show that, even with access to baseline methods and executable environments, current agents struggle to consistently match or improve upon baseline performance. These findings highlight persistent limitations in research-agent performance across real research tasks and point to opportunities for further improvement. We would like to publicly release the complete CSForge pipeline and the resulting environment dataset to support further research.

1 INTRODUCTION

Advances in search (Tongyi DeepResearch Team et al., 2025; MiroMind Team et al., 2026), reasoning (Guo et al., 2025; Jaech et al., 2024), and coding (Zeng et al., 2026; Kimi Team et al., 2025) have motivated growing interest in using large language model (LLM) agents for end-to-end computer science research (CSR). These CSR agents aim to integrate information gathering, method design, experimental execution, and iterative refinement within a single research workflow or system. Completing such long-horizon tasks requires coordinating these capabilities across an extended sequence of decisions and actions.

Recent efforts to build and improve CSR agents (Qian et al., 2026; Yamada et al., 2025; Yang et al., 2026) draw on human research experience and engineering heuristics, but focus primarily on agent scaffolding and continue to rely on manual iteration by researchers. Systematic progress requires *reliable* and *scalable* environments in which agent behavior can be measured, compared, and ultimately optimized (Chen et al., 2026). We consider two aspects of reliability: task specifications that faithfully represent the intended research problems and measurements that remain consistent across repeated runs under a fixed protocol. Scalability concerns the ability to construct large numbers of such environments with limited manual effort.

Research already places strong emphasis on evaluation, with studies often specifying datasets, metrics, and baselines. However, the reported measurements alone are insufficient to judge an agent's research outcomes. Building executable environments for CSR agents from these research resources

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.

requires reconstructing the underlying scientific research problem, establishing a fixed experimental protocol, designing reliable evaluation procedures, reproducing appropriate baselines, and defining meaningful improvement thresholds. Existing environments address these challenges through proxy tasks (Bragg et al., 2026; Xiao et al., 2025; Kon et al., 2026), LLM-based assessment (Chen et al., 2025; Wang et al., 2026), or manual construction (Zhang et al., 2025; Lupidi et al., 2026; Lyu et al., 2026; Nathani et al., 2025). These approaches provide useful artifacts yet raise different considerations for evaluation reliability and construction scalability.

Recent advances in agentic coding systems (Anthropic, 2025; OpenAI, 2025) make it increasingly feasible to automate the engineering-intensive parts of this process, such as inspecting research codebases, resolving dependencies, and reproducing baselines. These capabilities can substantially reduce the manual effort required to turn released artifacts into executable environments while preserving the paper's verifiable evaluation signals. Coding capability alone, however, is insufficient. Building a verifiable research environment also requires a clear scientific problem, a controlled experimental protocol, and explicit criteria for assessing progress. Automation must therefore go beyond software engineering and draw on human research experience to systematically reconstruct the scientific research process in reality.

To fill this gap, we introduce **CSForge**, an automated pipeline that transforms published research papers and their released artifacts into verifiable environments for CSR agents. CSForge incorporates human research experience through iterative pipeline design. It organizes the paper-to-environment transformation into three stages: *paper filtering* selects papers that support feasible experiments and objective evaluation; *task synthesis* reconstructs the scientific problem and its evaluation protocol; and *environment deployment* turns the resulting specification into executable, controlled experiments with reproduced baselines. Throughout the pipeline, evaluation is grounded in the source paper and its released artifacts, while LLM agents automate environment construction, validation, and repair.

We implement CSForge, apply it to a corpus of 16962 main track papers from 13 top-tier conferences spanning seven CSRankings subfields, and assess the results through automated checks and human review. Our analysis shows that the CSForge pipeline effectively identifies suitable papers, improves the reliability of evaluation protocols, and produces reproducible experimental environments. Overall, 80.0% of sampled filter-passing papers yield CSR environments that pass human quality review. We also manually inspect synthesis failures to identify unexpected issues arising from limitations in the current author agent's capabilities. These findings point to ways to improve the automation of CSR environment synthesis.

As a downstream application, we sample a 49-task test set from the synthesized environments and benchmark six frontier models. Even with access to the baseline methods, the best-performing model, Claude Opus 5, scores only 70.8 when asked to match or outperform them; its score drops to 38.9 when at least a 5% relative improvement is required. The models exhibit distinct performance profiles across research subfields and improvement thresholds, highlighting the difficulty and discriminative power of the synthesized environments.

In summary, we make the following contributions:

- We introduce CSForge, an autonomous pipeline that transforms published research papers into verifiable CSR environments, grounding evaluation in real research artifacts while automating the entire construction process.
- We systematically validate the pipeline: human review confirms a high rate of valid tasks and reproducible environments, while statistical analysis demonstrates the effect on improving the quality and efficiency of our design.
- We benchmark frontier models on a subset of synthesized environments, illustrating the potential of CSForge to provide environments that support the development of CSR agents.

2 Related Work

CSR Agents. LLM-based CSR agents have progressed from literature review and idea generation (Baek et al., 2025) to workflows integrating method development, experimentation, and evaluation. AI Scientist (Lu et al., 2024), CycleResearcher (Weng et al., 2025), and AIGS (Liu et al., 2024) automate research through structured pipelines, while AI Scientist-v2 (Yamada et al., 2025) incor-

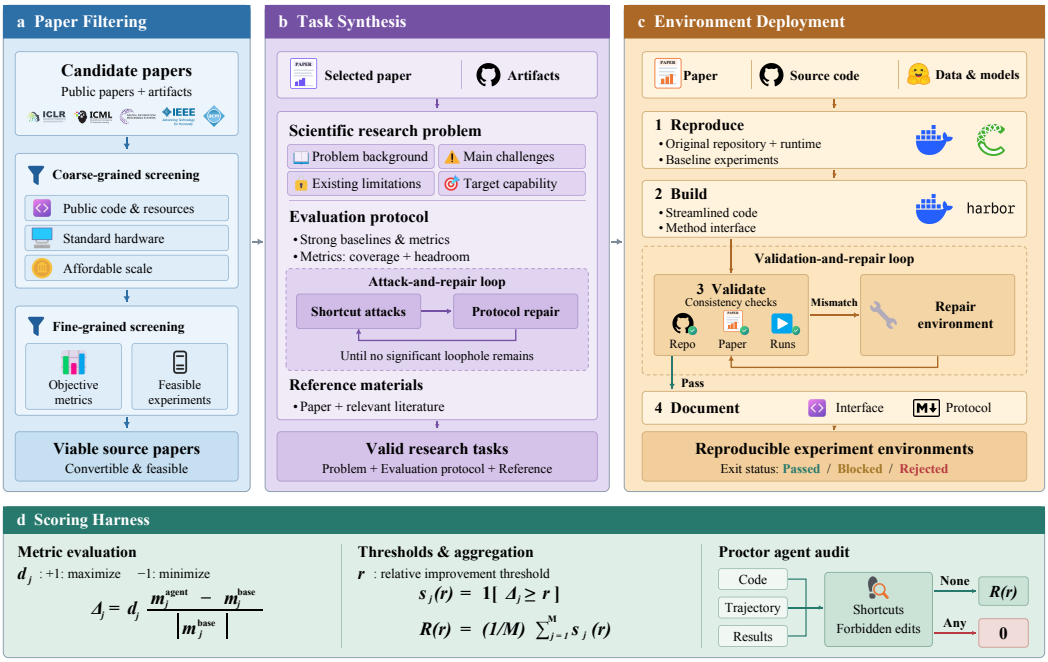


Figure 1: The CSForge environment construction pipeline consists of three stages: paper filtering, task synthesis, and environment deployment, all of which are carried out by the author agent. A scoring harness evaluates the solver agent's output through interaction with the environment, under the supervision of the proctor agent.

porates agentic search and iterative experimentation. AutoSci (Qian et al., 2026) and ARIS (Yang et al., 2026) extend general-purpose coding agents such as Claude Code and Codex with research-specific memory, tools, and skills. These advances underscore the need for reliable and scalable environments to systematically evaluate and improve CSR agents.

Benchmarks and Environments for CSR Agents. CORE-Bench (Siegel et al., 2024), CSR-Bench (Xiao et al., 2025), ResearchCodeBench (Hua et al., 2025), and EXP-Bench (Kon et al., 2026) assess computational reproducibility, repository deployment, research-code implementation, and experimental execution, respectively. PaperBench (Starace et al., 2025) evaluates the replication of AI research papers from scratch using hierarchical rubrics and an LLM-based judge. These benchmarks primarily assess reproduction or implementation rather than the discovery and validation of improved methods.

AstaBench (Bragg et al., 2026), MLR-Bench (Chen et al., 2025), and FIRE-Bench (Wang et al., 2026) target open-ended research problem solving and discovery, but rely largely on LLM-based evaluation. In contrast, MLRC-Bench (Zhang et al., 2025), AIRS-Bench (Lupidi et al., 2026), MLS-Bench (Lyu et al., 2026), and MLGym (Nathani et al., 2025) provide research environments in which agent-developed methods are executed and evaluated using objective metrics. However, their reliance on manual task curation and environment engineering limits scalability. CSForge complements these efforts by automating the construction of verifiable, end-to-end CSR environments from existing research artifacts.

3 The Design of CSForge

As shown in Figure 1, the CSForge pipeline automatically generates CSR environments, each pairing a verifiable research task with a reproducible experimental environment. We first formalize the environment structure and evaluation procedure (Section 3.1), then describe the construction pipeline (Sections 3.2–3.4).

3.1 ANATOMY OF A CSR ENVIRONMENT

Agent Roles. Our design involves three agent roles. The *author agent* executes the CSForge pipeline to construct experimental environments. The *solver agent* is evaluated on its ability to solve a CSR task within a synthesized environment. The *proctor agent* audits the solver agent's solutions for rule violations, as detailed below.

CSR Environments. A CSR environment is represented by the tuple:

$$E = \left(P,\ \mathcal{R},\ \mathcal{C},\ \{(m_j^{\text{base}}, d_j)\}_{j=1}^{M},\ H\right), \tag{1}$$

whose components are defined as follows:

- **Problem statement** P: a method-agnostic specification of the scientific problem, with detailed experimental requirements and implementation documentation, including the metrics $\{m_j\}_{j=1}^{M}$ to be reported and the required result format.
- **Reference materials** $\mathcal{R}$: the source paper and a curated set of related references that supply the scientific context, technical definitions, and experimental details of the task.
- **Execution environment** $\mathcal{C}$: a containerized, reproducible experimental environment comprising a streamlined codebase, experimental resources (e.g., datasets and models), Docker images, reference implementations, and a documented interface for integrating new methods.
- **Baseline results** $\{(m_j^{\text{base}}, d_j)\}_{j=1}^{M}$: for each metric m_j, a standardized baseline value m_j^{base} reproduced from the source paper and its optimization direction $d_j \in \{+1, -1\}$, where $d_j = 1$ indicates that higher values are better and $d_j = -1$ that lower values are better.
- **Evaluation harness** H: a standardized scoring program that parses the agent's reported results, checks them against the executed experiments, and computes the final reward.

Observability. The components of E are partitioned into a solver-facing part and a held-out part. The solver agent observes only $(P, \mathcal{R}, \mathcal{C})$: it receives the problem statement and reference materials, and interacts freely with the execution environment to develop and run its method. In contrast, the baseline values $\{m_j^{\text{base}}\}$ and the evaluation harness H are withheld from the solver agent.[1] The proctor agent has access to all components.

Evaluation. After the solver completes its work, the evaluation harness H executes the proposed method and reports the resulting metric values $\{m_j^{\text{agent}}\}_{j=1}^{M}$. It then scores each metric by comparing the solver agent's result with the hidden baseline at an improvement threshold $r \geq 0$:

$$s_j(r) = \begin{cases} 1, & \text{if } d_j \cdot \frac{m_j^{\text{agent}} - m_j^{\text{base}}}{|m_j^{\text{base}}|} \geq r, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Thus, $r = 0$ requires the agent to match or outperform the baseline, while $r = 0.05$ requires at least a 5% relative improvement. The reward is initially computed as the arithmetic mean of the scores across all M metrics:

$$R(r) = \frac{1}{M} \sum_{j=1}^{M} s_j(r). \tag{3}$$

In parallel with metric scoring, the proctor agent reviews the solver agent's execution trajectory and submitted code for compliance with the task requirements, including prohibitions on shortcuts and unauthorized code modifications. If any noncompliant behavior is detected, the reward is set to $R(r) = 0$, regardless of the reported metric values.

3.2 PAPER FILTERING: IDENTIFYING VIABLE SOURCE MATERIALS

CSForge starts from published CS papers and identifies those suitable for environment construction before proceeding to the more token-intensive stages.

[1] The metric definitions and their optimization directions are disclosed in P; only the concrete baseline values and the scoring logic are hidden.

To support efficient construction of a CSR environment, a paper must satisfy two complementary requirements. First, the paper and its released artifacts must provide sufficient support for constructing an executable CSR task and environment. Second, the resulting task should faithfully capture the core research problem while remaining feasible to execute within a reasonable resource budget. The first requirement can be assessed from artifact availability and the paper's experimental setup, whereas the second requires a deeper understanding of the research workflow and the underlying problem. We therefore use a two-stage, coarse-to-fine screening procedure.

- **Coarse-grained screening**. This stage filters out papers that lack the code or other resources needed for experimentation, depend on specialized hardware such as FPGAs, robots, or sensors, or require large-scale computing infrastructure.
- **Fine-grained screening**. This stage further excludes studies outside the scope of the current task format, including theoretical analyses, profiling studies, and large-scale system implementations that do not readily support the prescribed metric-based evaluation. It also removes papers whose experiments require excessive computational resources or engineering effort and therefore cannot be validated within a practical time budget.

Because these criteria involve a degree of subjectivity, it is difficult to assess paper suitability with complete accuracy from the paper alone. In our design, the primary goal of screening is to reduce the resources spent on costly downstream attempts for unsuitable papers. We therefore prioritize the viability of papers that pass the filter and adopt a relatively conservative screening strategy, even if this may exclude some potentially suitable papers.

3.3 Task Synthesis: Designing Valid CSR Tasks

The task synthesis stage defines the core problem $(P, \mathcal{R})$ and evaluation protocol H of a CSR environment, thereby determining whether the resulting environment constitutes a valid and meaningful research task. This stage therefore draws more deeply on human research experience to guide problem formulation and evaluation design.

Scientific Problem Formulation. First, the author agent reconstructs a scientific problem from the source paper. The problem should leave room for novel solutions without prescribing a particular method, while remaining sufficiently constrained to ensure fair and meaningful comparisons. To constrain this reformulation, we use a structured problem specification. The specification comprises four components: (i) *problem background*, which introduces the scientific context; (ii) *main challenges* and (iii) *existing limitations*, which identify the key unresolved issues and opportunities for improvement; and (iv) *target capability*, which defines the desired properties and constraints of a candidate solution. When specifying these requirements, we use the source paper's method as a reference to preserve comparable assumptions, inputs, outputs, and experimental constraints without prescribing that method as the solver's solution.

Evaluation Protocol Design. Once the scientific problem is formulated, the author agent constructs an evaluation protocol that assesses the target capabilities through concrete experiments. Drawing on the main empirical studies in the source paper, the protocol specifies the evaluation settings, baselines, metrics, and experimental requirements for assessing candidate methods. We use strong baselines, typically including the method proposed in the source paper and, when appropriate, other competitive methods evaluated under comparable settings. The protocol is designed around two criteria: *comprehensive coverage*, meaning that the experiments and metrics collectively capture the essential dimensions of the target capabilities; and *improvement headroom*, meaning that sufficient room for improvement remains beyond existing baselines to measure meaningful progress.

To improve coverage, we design an iterative attack-and-repair procedure. The author agent attempts to construct naive shortcut methods that exploit weaknesses in the current evaluation and achieve high scores without genuinely addressing the intended problem. When such a shortcut is found, the protocol is strengthened with additional metrics, constraints, or experimental conditions, and the process repeats until no significant loophole is identified. To assess improvement headroom, we compare the strongest available baseline against a theoretical oracle and retain metrics for which the remaining gap is sufficiently large to support meaningful differentiation.

Reference Material Curation. Finally, the author agent assembles the source paper and the references most relevant to the background, the definition of the scientific problem and its evaluation protocol. These materials are organized into a standardized format for the reference of solver agents and provided as the task's reference set $\mathcal{R}$, supplying the scientific context, technical definitions, and experimental details needed to understand and address the problem.

3.4 Environment Deployment: Constructing Reproducible Environments

After synthesizing a valid CSR task, CSForge implements its evaluation protocol in a standardized experimental environment, yielding the remaining components $\mathcal{C}$, $\{(m_j^{\text{base}}, d_j)\}$, and H of the environment tuple. This stage aims to ensure reproducibility: executing the code under a predefined protocol should yield verifiable results that reproduce the empirical findings of the selected experiments in the source paper.

To ensure executability, we adopt a reproduce-then-build strategy. The author agent first retrieves the code and runtime resources released with the source paper and reproduces the baseline experiments selected during task synthesis using the original repository. Once the target experiments run successfully, the agent extracts the task-relevant components into a streamlined, independent codebase. It then packages the required dependencies and resources, implements the evaluation harness, and defines a common interface for integrating new methods.

Quality Control. After constructing the streamlined environment, we apply an iterative validation-and-repair loop to ensure its fidelity and reproducibility. The author agent repeatedly executes the target experiments and checks three forms of consistency: (1) *implementation consistency*, whether the streamlined environment reproduces the results obtained from the original repository; (2) *claim consistency*, whether the reproduced results support the corresponding empirical claims in the source paper; and (3) *run-to-run consistency*, whether repeated executions yield stable results.

Exception Handling. Despite prior screening and iterative quality control, deployment may still encounter unexpected obstacles, such as incomplete released artifacts, model or data downloads that require authentication, or experimental results that cannot be reproduced. We therefore equip the author agent with an explicit exception-handling mechanism. Each deployment terminates in one of three states. PASSED indicates that construction and quality-control checks have completed successfully. BLOCKED indicates a potentially recoverable issue that requires external intervention, such as authentication requirements. REJECTED indicates a fundamental obstacle that prevents reliable deployment, such as missing essential artifacts or persistent failure to reproduce the experiment.

For each environment with PASSED status, the author agent documents the method interface and evaluation protocol so that solver agents can develop and evaluate new methods without reconstructing the underlying experimental setup.

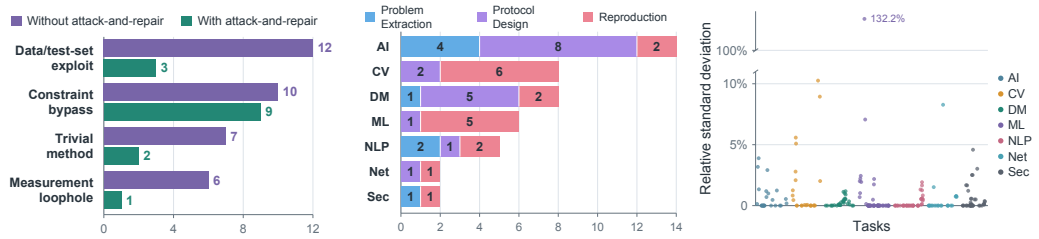
4 Implementing and Evaluating CSForge

We implement the pipeline described in Section 3 using a frontier agentic coding system and evaluate its outputs at each stage. This evaluation examines how effectively current models and agent systems can carry out the CSForge pipeline in practice. We first describe the implementation details in Section 4.1, and then analyze the quality of each pipeline component in Section 4.2.

4.1 Implementation of the Pipeline

We implement each pipeline stage as a modular skill for Claude Code and combine these skills into an end-to-end construction pipeline. The author agent invokes the corresponding skill at each stage.

For *paper filtering*, coarse-grained screening combines agent-based information extraction with rule-based verification. Given the target server configuration, the agent extracts information about artifact availability, hardware requirements, and experimental scale. A rule-based script then uses this information to determine whether the paper meets the screening criteria. For fine-grained screening, the agent applies a predefined rubric, checking each criterion before making a final decision. The complete rubric is provided in the Appendix A.1.



(a) (b) (c)

Figure 3: Evaluation of the CSForge pipeline. (a) Shortcut behaviors with and without attack-and-repair. (b) Issues identified during expert review, grouped by subfield. (c) Variation in baseline measurements across repeated runs.

For *task synthesis*, we provide the author agent with the structured scientific-problem template and extraction guidelines described in Section 3.3, together with script-based format checks. The agent designs an initial evaluation protocol based on the source paper's main experiments, then enters the attack-and-repair loop. In each round, three subagents independently search for shortcut solutions and report the exploited metrics or weaknesses to the main author agent. The author agent aggregates these findings, determines whether the protocol remains vulnerable, and revises it accordingly. The task is discarded if significant loopholes remain after five rounds.

For *environment deployment*, the author agent uses Docker to provide a consistent execution platform and initializes the environment using the dependency and configuration files released with the source repository. It then iteratively repairs and extends the environment while reproducing the target experiments. After construction, the agent runs the consistency-check and repair loop described above, with a limit of five rounds. The agent may terminate at any point with BLOCKED or REJECTED status if the corresponding exception condition is met. Successful completion results in PASSED status. Before termination, automated scripts record the execution status and run additional integrity checks for downstream analysis. The successful outputs are organized in harbor format.

In our experiments, the author agent is powered by Claude Opus 4.7 during task synthesis. At official API rates, the average cost is $10.67 per successfully synthesized task. Prompt templates and skill specifications are provided in the Appendix A.

4.2 EVALUATION OF THE PIPELINE

Following the implementation of CSForge using Claude Code, our next question is: how effectively does it carry out the design of CSForge? This evaluation examines the ability of current models and agent systems to serve as author agents for CSR environment construction. We apply this implementation to main-track papers published in 2024–2025 at 13 top-tier conferences across seven CSRankings subfields. We then evaluate the author agent's performance through human review and statistical analysis of the construction results.

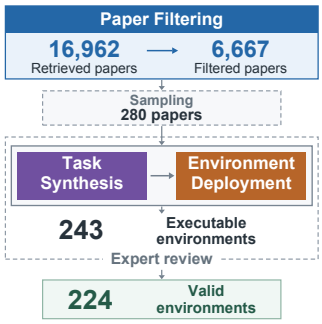


Figure 2: The procedure of env construction and validation.

Paper Filtering. As shown in Figure 2, paper filtering retains 6,667 of the 16,962 collected papers (39.3%). Of 280 sampled papers, 243 yield executable environments; 224 of these pass expert review and are classified as valid CSR environments. Construction failures mainly stem from missing essential code or data in released repositories, or from an inability to reproduce the reported findings. Environments rejected during expert review are discussed in the following paragraphs.

To assess agreement with human judgments, we select 70 papers labeled suitable and 70 unsuitable by experts applying the same rubric as the author agent. The agent correctly rejects 68 of the 70 un-

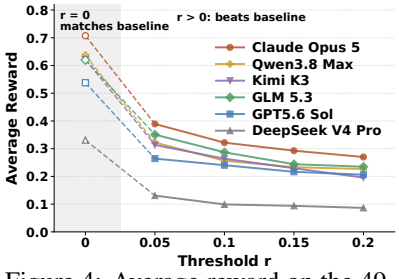


Figure 4: Average reward on the 49-task subset under different r.

Table 1: Reward scores by subfield on the 49-task evaluation subset at $r = 0.05$. Scores are reported on a 0–100 scale. Each subfield contains seven tasks. The highest score in each column is shown in bold.

Model	AI	CV	DM	ML	Net	NLP	Sec	Avg
Claude Opus 5	39.7	**31.0**	**14.3**	**43.8**	**46.4**	51.2	**46.2**	**38.9**
GLM 5.3	**57.5**	21.4	4.8	30.6	44.1	45.2	41.9	35.1
Kimi K3	39.7	21.4	4.8	39.0	36.9	47.6	30.0	31.4
Qwen3.8 Max	29.0	16.7	8.3	39.0	36.9	**52.4**	43.3	32.2
GPT-5.6 Sol	29.0	4.8	9.5	41.8	29.8	48.8	21.4	26.4
DeepSeek V4 Pro	18.3	4.8	4.8	7.7	13.1	33.3	9.8	13.1

suitable (97% recall), but incorrectly rejects 9 suitable papers. Most erroneous rejections stem from inaccurate estimates of experimental scale and resource requirements, whereas erroneous acceptances mainly from treating validation experiments in theoretical papers as algorithmic experiments.

Task Synthesis. Figure 3b summarizes the issues identified during expert review. The main failures in task synthesis are invalid problem formulations that depart from the prescribed template and evaluation protocols that do not align with the source paper's main objectives. Manual inspection of author-agent trajectories suggests two main causes: failures to follow instructions in very long contexts and difficulty interpreting results presented in figures. The latter biases experiment selection toward results reported in tables. These issues are particularly pronounced in AI and data mining.

For each of the same 49 source papers, we construct two environment variants, with and without attack-and-repair. Evaluating both variants using six models yields 588 solution trajectories (294 per variant). The proctor agent audits the execution trajectories and submitted code for potential shortcut behaviors. As shown in Figure 3a, attack-and-repair strengthens evaluation protocols and reduces manipulation of data or test sets, method shortcuts, and exploitation of measurement loopholes. However, its effect on violations of method requirements stated in natural language is limited, so proctor oversight remains necessary.

Environment Deployment. When the required resources are available and dependencies are manageable, deployment generally succeeds, but the resulting environments still need to be checked for quality. The Reproduction category in Figure 3b covers cases in which reproduced results differ substantially from those reported in the source paper or fail to support its claims. These failures most often arise when, under time and cost constraints, the author agent uses smaller datasets or models or simplifies experimental settings.

To assess run-to-run consistency, we repeat each environment's baseline experiments execution three times(note that each execution could include multiple underlying specified by author agent) and compute the relative standard deviation of each metric. As shown in Figure 3c, most measured metrics have relative standard deviations no greater than 5%. A few outliers show greater variation, which we attribute to the experimental simplifications described above.

Overall Usability and Remaining Gap. Taken together, these results suggest that current frontier models, when paired with agentic coding systems, can serve as useful, though not yet fully autonomous, author agents for CSR environment construction. Among the sampled papers that pass our viability filter, 86.8% yield executable environments, and 80.0% yield environments that pass human quality review. Thus, the current implementation completes end-to-end construction of a valid environment for roughly four out of five sampled papers that pass the filter. Several recurring failure modes remain: errors in identifying the core research problem and representative experiments; difficulty inferring implicit experimental requirements, such as computational scale; and failures to faithfully reproduce or correctly simplify complex research artifacts during deployment. These errors indicate that the main bottleneck is no longer basic code generation, but the reliable integration of scientific understanding, experimental design, and long-horizon artifact engineering. Closing this gap will require author agents that can interpret evidence across text, tables, and figures more accurately, preserve experimental assumptions more faithfully, and recover from reproduction failures with less external intervention.

5 BENCHMARKING WITH CSFORGE

As a downstream application of CSForge, we use the synthesized environments to benchmark CSR agents on their ability to develop competitive research methods. This evaluation complements the construction-quality assessment by testing the practical utility of the resulting environments.

Experimental Setup. We benchmark solver agents powered by six frontier models using a stratified random sample of 49 human-validated environments, with seven tasks from each of the seven CS subfields. The reseoning effort of all models are set to high. GPT-5.6 Sol is tested with its official Codex scaffold while all other models are tested with Claude Code.

Agents receive baseline methods and executable environments and are asked to develop and evaluate candidate methods. Following Section 3.1, we evaluate their outputs under different relative improvement thresholds r. At $r = 0$, the objective is to produce a method that matches or outperforms the baselines *given access to those baseline methods*, rather than reproduce an unknown method from scratch. At $r = 0.05$, credit requires at least a 5% relative improvement in the designated optimization direction. A metric receives credit only if the improvement threshold is met relative to the strongest reference baseline for that metric. We average scores over metrics within each task and then over tasks, reporting results on a 0–100 scale in the text and table. Submissions with missing result files or rule violations detected by proctor agent receive zero reward.

Results. Figure 4 summarizes performance under different improvement thresholds. At $r = 0$, Claude Opus 5 achieves the highest score of 70.8, followed by Qwen3.8 Max at 63.7, Kimi K3 at 62.2, and GLM 5.3 at 62.0. GPT-5.6 Sol and DeepSeek V4 Pro score 53.8 and 33.1, respectively. Thus, even with access to baseline methods and executable environments, the evaluated agents do not consistently outperform or at least match baseline performance across the required metrics.

At the primary threshold $r = 0.05$ which requires a 5% improvement, the scores substantially become lowers in all six configurations. Claude Opus 5 retains the highest score at 38.9, while the other configurations score between 13.1 and 35.1. The decrease from $r = 0$ to $r = 0.05$ shows that many outcomes credited for matching the baseline fall short of a 5% improvement. The ranking also changes: GLM 5.3 moves from fourth at $r = 0$ to second at $r = 0.05$, scoring 35.1 compared with 32.2 for Qwen3.8 Max and 31.4 for Kimi K3. In this evaluation, stronger baseline-matching performance does not necessarily translate into better performance at a higher improvement threshold. Reporting both thresholds captures this distinction.

Table 1 further reveals variation across research subfields. At $r = 0.05$, Claude Opus 5 leads in five of the seven subfields, but does not dominate uniformly: GLM 5.3 achieves the highest AI score of 57.5, compared with 39.7 for Claude Opus 5, while Qwen3.8 Max records the highest NLP score of 52.4. Performance also varies considerably across the sampled research problems. Scores on DM tasks range from only 4.8 to 14.3 across the six agents, whereas NLP scores range from 33.3 to 52.4. These differences reveal performance patterns that are obscured by the aggregate ranking, although the small number of tasks per subfield warrants caution in generalizing beyond this subset.

Together with the validity and reproducibility checks in Section 4, these results suggest that CSForge produces environments that are challenging and discriminative for the evaluated agent configurations. The benchmark demonstrates how the synthesized environments can compare agents' baseline-matching and baseline-improvement performance across research settings.

6 CONCLUSION

We present **CSForge**, an autonomous pipeline for constructing verifiable CSR environments from published research artifacts. Through paper filtering, task synthesis, and environment deployment, CSForge identifies viable research problems, formulates valid tasks with room for improvement, and provides reproducible execution environments with standardized baselines and evaluation protocols. Our evaluation shows that CSForge achieves high construction success and human-review pass rates while substantially reducing manual effort. The resulting tasks remain challenging and discriminative for frontier agents. Experiments on the 49-task evaluation subset further demonstrate the utility of CSForge for benchmarking and analyzing CSR agents.

7 AI USE STATEMENT

We used generative AI tools to support environment construction, testing, and monitoring in our experiments. We also used large language models (LLMs), including GPT-6 and Claude Opus 5, to improve the clarity and readability of the manuscript. We have reviewed all AI-assisted work and take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the assistance of generative AI.

REFERENCES

Anthropic. Claude 3.7 Sonnet and Claude Code. `https://www.anthropic.com/news/claude-3-7-sonnet`, February 2025. Accessed: 2026-09-15.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative research idea generation over scientific literature with large language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

Jonathan Bragg, Mike D'Arcy, Nishant Balepur, Dan Bareket, Bhavana Dalvi Mishra, Sergey Feldman, Dany Haddad, Jena D. Hwang, Peter Jansen, Varsha Kishore, Bodhisattwa Prasad Majumder, Aakanksha Naik, Sigal Rahamimov, Kyle Richardson, Amanpreet Singh, Harshit Surana, Aryeh Tiktinsky, Rosni Vasu, Guy Wiener, Chloe Anastasiades, Stefanus Candra, Jason Dunkelberger, Daniel Emery, Rob Evans, Malachi Hamada, Regan Huff, Rodney Kinney, Matt Latzke, Jaron Lochner, Ruben Lozano-Aguilera, Ngoc-Uyen Nguyen, Smita Rao, Amber Tanaka, Brooke Vlahos, Peter Clark, Doug Downey, Yoav Goldberg, Ashish Sabharwal, and Daniel S. Weld. AstaBench: Rigorous benchmarking of AI agents with a scientific research suite. In *The Fourteenth International Conference on Learning Representations*, 2026.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. MLR-Bench: Evaluating AI agents on open-ended machine learning research. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. Datasets and Benchmarks Track.

Mouxiang Chen, Lei Zhang, Yunlong Feng, Xuwu Wang, Wenting Zhao, Ruisheng Cao, Jiaxi Yang, Jiawei Chen, Mingze Li, Zeyao Ma, Hao Ge, Zongmeng Zhang, Zeyu Cui, Dayiheng Liu, Jingren Zhou, Jianling Sun, Junyang Lin, and Binyuan Hui. SWE-Universe: Scale real-world verifiable environments to millions, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. ResearchCodeBench: Benchmarking LLMs on implementing novel machine learning research code. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. Datasets and Benchmarks Track.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Patrick Tser Jern Kon, Qiuyi Ding, Jiachen Liu, Xinyi Zhu, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, Matei Zaharia, and Ang Chen. EXP-Bench: Can AI conduct AI research experiments? In *The Fourteenth International Conference on Learning Representations*, 2026.

Zijun Liu, Kaiming Liu, Yiqi Zhu, Xuanyu Lei, Zonghan Yang, Zhenhe Zhang, Peng Li, and Yang Liu. AIGS: Generating science from AI-powered automated falsification, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery, 2024.

Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, Jean-Christophe Gagnon-Audet, Chee Hau Leow, Sandra Lefdal, Hossam Mossalam, Abhinav Moudgil, Saba Nazir, Emanuel Tewolde, Isabel Urrego, Jordi Armengol Estape, Amar Budhiraja, Gaurav Chaurasia, Abhishek Charnalia, Derek Dunfield, Karen Hambardzumyan, Daniel Izcovich, Martin Josifoski, Ishita Mediratta, Kelvin Niu, Parth Pathak, Michael Shvartsman, Edan Toledo, Anton Protopopov, Roberta Raileanu, Alexander Miller, Tatiana Shavrina, Jakob Foerster, and Yoram Bachrach. AIRS-Bench: A suite of tasks for frontier AI research science agents, 2026.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan-ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI, 2026.

MiroMind Team, S Bai, L Bing, L Lei, R Li, X Li, X Lin, E Min, L Su, B Wang, et al. MiroThinker-1.7 & H1: Towards heavy-duty research agents via verification. *arXiv preprint arXiv:2603.15726*, 2026.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. In *Second Conference on Language Modeling*, 2025.

OpenAI. Introducing Codex. `https://openai.com/index/introducing-codex/`, May 2025. Accessed: 2026-09-15.

Weitong Qian, Beicheng Xu, Zhongao Xie, Bowen Fan, Guozheng Tang, Jiale Chen, Xinzhe Wu, Mingtian Yang, Chenyang Di, Jiajun Li, Lingching Tung, Peichao Lai, Yifei Xia, Ziyi Guo, Yanwei Xu, Yanzhao Qin, Shaoduo Gan, Xupeng Miao, and Bin Cui. AutoSci: A memory-centric agentic system for the full scientific research lifecycle, 2026.

Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-Bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi DeepResearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Zhen Wang, Fan Bai, Zhongyan Luo, Jinyan Su, Kaiser Sun, Xinle Yu, Jieyuan Liu, Kun Zhou, Claire Cardie, Mark Dredze, Zhiting Hu, and Eric P. Xing. FIRE-Bench: Evaluating AI agents on the rediscovery of scientific insights. In *Forty-third International Conference on Machine Learning*, 2026.

Yixuan Weng, Minjun Zhu, Guangsheng Bao, Hongbo Zhang, Jindong Wang, Yue Zhang, and Linyi Yang. CycleResearcher: Improving automated research via automated review. In *International Conference on Learning Representations*, volume 2025, pp. 3669–3709, 2025.

Yijia Xiao, Runhui Wang, Luyang Kong, Davor Golac, and Wei Wang. CSR-Bench: Benchmarking LLM agents in deployment of computer science research repositories. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 12705–12723, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.633.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist-v2: Workshop-level automated scientific discovery via agentic tree search, 2025.

Ruofeng Yang, Yongcan Li, and Shuai Li. ARIS: Autonomous research via adversarial multi-agent collaboration. *arXiv preprint arXiv:2605.03042*, 2026.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant D. Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. MLRC-Bench: Can language agents solve machine learning research challenges? In *Advances in Neural Information Processing Systems*, volume 38, 2025. Datasets and Benchmarks Track.

A Rubrics and Templates in the Implementation

For readability, the rubrics and prompts in this appendix are simplified presentations of the operational instructions. We retain the core selection criteria and design principles while omitting output-formatting details, edge-case handling and associated constraints, and selected references to auxiliary scripts.

A.1 Rubrics for Paper Filtering

Paper filtering proceeds in two stages. Coarse-grained screening extracts resource and experiment information for rule-based feasibility checks. Fine-grained screening assesses whether the paper supports a meaningful and practically executable CSR task.

Coarse-grained screening: information extraction

Read the paper and extract the following information, focusing on its main experiments.

1. **Basic information.** Record the title and abstract.
2. **Artifact availability.** Determine whether the code, data, and other resources required to reproduce the experiments are publicly available. Record the released artifacts, access links, and any missing resources.
3. **Hardware requirements.** Identify requirements that the reference server cannot satisfy, such as FPGAs, TPUs, sensors, or robots. Flag experiments requiring more than two server nodes. Distinguish specialized hardware from standard GPU computing requirements.
4. **Experimental scale.** Identify large-scale model training, large-scale simulations, and profiling across multiple servers.
5. **Type of work.** Determine whether the paper introduces a dataset, benchmark, or simulator; proposes an algorithm or system for an existing problem; or represents another type of contribution. State the problem being addressed.
6. **Evaluation components.** List the datasets, benchmarks, simulators, baseline methods, and metrics used in the main experiments. For a new benchmark or dataset, identify the metrics used to evaluate methods on it.

Hardware reference. Each reference server has two Intel Xeon Platinum 8260 CPUs, eight NVIDIA A100 GPUs with 80 GB memory each and NVLink, 512 GB RAM, two 1 TB NVMe SSDs, two 10 TB HDDs, and 100 Gbps Ethernet.

Evidence and uncertainty. Support important findings with paper sections, tables, short quotations, or artifact links. Record insufficiently supported findings as *unknown*; do not infer availability or feasibility from missing information.

Fine-grained screening: suitability assessment

Read the paper together with the extracted information. Assess whether it can support a research task with a clear scientific challenge, accessible experimental resources, and objective evaluation. Apply the following criteria to the main contribution and experiments.

1. **System complexity.** A contribution centered on implementing a large, complex system may be unsuitable if its implementation burden prevents practical task construction.
2. **Environment complexity.** Assess whether reproducing the experiments requires intricate kernel modifications or similarly demanding engineering that makes environment construction impractical.
3. **Research relevance.** Exclude profiling tools, experimentation suites, or service platforms that do not provide a suitable task for evaluating scientific research capabilities.
4. **Benchmark and dataset contributions.** Retain such papers only when they define a clear scientific challenge and provide well-defined baselines.
5. **Empirical comparability.** Exclude purely theoretical work and studies whose experiments provide analysis without directly comparable algorithmic or system performance.
6. **Objective evaluation.** The main evaluation must use objective numerical metrics. Exclude papers whose primary evaluation relies on human assessment or an LLM judge.

7. **Resource dependence.** Exclude methods tightly coupled to an unavailable model API or other resource when no viable substitute exists.

Provide a suitability decision and a concise rationale grounded in the paper's main research problem and experimental requirements.

A.2 Prompts for Scientific Problem Extraction

The author agent reconstructs a method-agnostic scientific problem before designing its evaluation protocol.

Scientific problem extraction

Reconstruct the scientific problem that the paper seeks to solve, rather than summarizing the paper or restating its proposed method. Anchor the formulation in the main claim, the main comparative experiments and tables, and their analysis. For algorithm or system papers, identify the problem being solved. For benchmark or dataset papers, identify the scientific challenge being evaluated.

Required components

1. **Problem background.** Describe the broader task or domain and explain why the problem matters.
2. **Main challenges.** Identify two to four concrete technical difficulties and the constraints that make the problem challenging.
3. **Existing limitations.** Identify two to four specific limitations of prior approaches or baselines that motivate the problem.
4. **Target capability.** State what a successful method should enable and the properties and constraints it must satisfy, independently of the paper's specific solution.

Formulation rules

- Do not prescribe the paper's method. Mention it only when necessary to explain limitations of existing approaches; retain the constraints inherent to the scientific problem.
- When the paper has multiple contributions, select the one most central to its main result. Do not substitute secondary ablations or implementation details for the core research problem.
- If the paper is primarily theoretical or analytical, or lacks a reproducible empirical core, mark it as unsuitable rather than forcing a benchmark task.
- Describe the intended scientific contribution. Do not explicitly formulate the objective as "improving metric performance" or "outperforming the baseline."

A.3 Prompts for Evaluation Protocol Design

Given the scientific problem, the author agent specifies experiments, metrics, baselines, and experimental conditions, then refines the protocol through iterative attack-and-repair.

Evaluation protocol design

Design an evaluation protocol that reveals the relative strengths and weaknesses of candidate methods for the scientific problem. Ground the protocol in the paper's main experiments and claims, and specify the conditions required for fair and meaningful comparisons.

Experiment design
Select experiments from the paper's main figures and tables, or combine complementary experiments that capture different aspects of the problem.

- **Comparability.** Evaluate results using objective numerical metrics. Avoid evaluations that rely on subjective interpretation or visualizations intended solely for human understanding.
- **Simplicity.** Favor direct measurements of method effectiveness aligned with the paper's main claims. Avoid elaborate ablation suites and indirect analysis experiments.

Metric selection

Select established, clearly defined metrics from the paper's main experiments. Avoid ambiguous or difficult-to-compare quantities, and apply the following principles:

- **Simplicity.** Avoid redundant metrics that measure essentially the same property.
- **Comprehensive coverage.** Cover the essential dimensions of the target capability so that a trivial method cannot obtain high scores by exploiting a single dimension while neglecting others.
- **Improvement headroom.** Retain metrics for which meaningful improvement over the baselines remains possible. Metrics used solely to verify constraints are exempt from this requirement.

Baseline selection
Choose baselines representing the strongest known methods for the problem, typically including the paper's proposed method. Include additional baselines from the paper when they outperform that method on some metrics, so that the protocol captures complementary strengths and tradeoffs. For benchmark or dataset papers that do not introduce a new method, select the strongest reported baseline.

Attack-and-repair
Within the configured subagent budget, iteratively ask subagents to search for shortcuts that exploit weaknesses in the evaluation, bypass constraints, or achieve high scores through trivial solutions. Validate the proposed shortcuts before treating them as loopholes. Repair confirmed loopholes by expanding metric coverage, tightening existing constraints, or adding necessary constraints, and reassess the revised protocol.

B Implementation Details

Evaluation subset sampling. We construct the 49-task evaluation subset from human-validated environments using stratified sampling, with seven tasks from each of the seven CS subfields. Before sampling, we test all environments with Claude Opus5 and DeepSeek V4 Pro and exclude tasks for which none of the evaluated models can complete experimental execution. We also exclude tasks with any metric exhibiting a relative standard deviation of at least 5% across repeated baseline runs. Stratified sampling is then performed on the remaining eligible tasks.

Proctor calibration. The proctor agent is powered by Claude Opus 5. We adjust and calibrate it using ten representative, human-labeled trajectories: five labeled as non-cheating and five labeled as cheating, with the latter representing different types of violations. The proctor is deployed for benchmark auditing once its judgments agree with the human annotations on all ten calibration trajectories.

Human review of environments. Three experts independently review each environment. Each expert assesses whether the environment meets all review criteria and marks it as invalid if any criterion is not satisfied. The final validity decision is determined by majority vote across the three experts. Multiple issues may be recorded for a single environment.