REXBENCH: Can coding agents autonomously implement AI research extensions?

Anonymous ACL submission

Abstract

Agents based on Large Language Models (LLMs) have shown promise for performing sophisticated software engineering tasks autonomously. In addition, there has been progress towards developing agents that can perform parts of the research pipeline in machine learning and the natural sciences. We argue that research *extension* and its implementation is a critical capability for such systems, and introduce **REXBENCH** to support the evaluation of this capability. **REXBENCH** is a benchmark consisting of realistic extensions of 12 research papers that aim to investigate *novel* research hypotheses. Each task is set up as an extension to an existing research paper and codebase, accompanied by domain expert-written instructions. **REXBENCH** is robust to data contamination, and supports an automatic evaluation infrastructure that executes agent outputs to determine whether the success criteria are met. We use this benchmark to evaluate 12 LLM agents implemented using two different frameworks: aider and OpenHands. We find that all agents fail to autonomously implement the majority of the extensions, with the best agent at around 33% success rate. Although the success rate improves with additional human-written hints, the best performance under this setting remains below 44%. This indicates that current agents are still short of being able to handle realistic research extension tasks without substantial human guidance.

1 Introduction

Interesting research necessarily builds on other research. In this regard, *extensions* of existing research are important starting points to new investigations, potentially building up towards exciting novel discoveries. In light of recent growing interest in building Large Language Model (LLM) agents that can conduct scientific research in an autonomous manner, we propose **REXBENCH**, a benchmark aiming to evaluate LLM agents' ability to extend existing AI research, with an initial focus on Natural Language Processing (NLP) and Machine Learning (ML). More specifically, **REXBENCH** tests whether LLM agents can autonomously implement research extension experiments via code in a hypothesis-guided manner (Luo et al., 2025), where the extension hypotheses are provided to the system as verbal instructions along with relevant background material including the research paper(s) and the corresponding codebase. Our benchmark consists of realistic extensions of 12 recently published research papers in the field, accompanied by domain expert-written extension instructions (see Appendix F for a sample task instruction). The extension tasks cover various aspects of implementation involving changes to the model, algorithm, data, and evaluation method. The main metric of success is numerical replication of the outcome of domain-expert implemented "gold" solutions for the extension task. We provide an automatic evaluation infrastructure to execute the LLM agent-implemented solutions and evaluate the outcomes. The executions of both the gold solutions and system solutions are conducted in virtual machines with exactly the same specifications to control for experimental variation. **REXBENCH** furthermore is robust to data contamination issues that affect the majority of existing benchmarks: the solutions and the success criteria for our extension tasks only exist in our held-out evaluation infrastructure and do not exist anywhere online.

We tested twelve agents based on an array of LLM backbones (Claude 4/3.7 Sonnet (Anthropic, 2025, 2024), GPT-5 (OpenAI, 2025), o1 (Jaech et al., 2024), o4-mini, and DeepSeek-R1 (Guo et al., 2025), using two different agent frameworks (aider and OpenHands). Many agents struggled on our benchmark, achieving success rates close to zero for most tasks. Agents with Claude 4/3.7 Sonnet and GPT-5 as backbone showed promise,

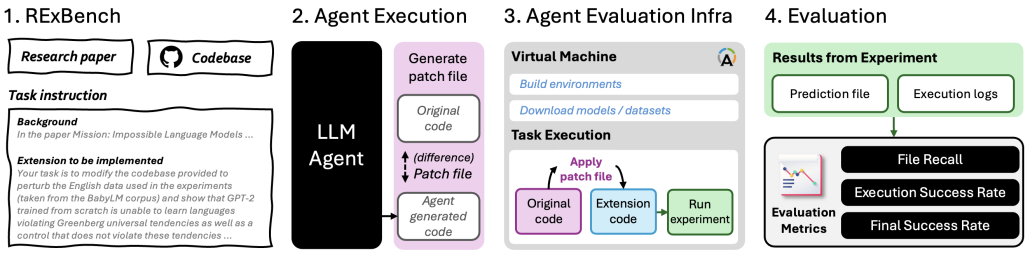

Figure 1: End-to-end workflow of **REXBENCH**: (1) An LLM agent receives inputs consisting of the research paper(s), the original codebase, and an extension instruction; (2) the system implements the extension and a patch file is obtained; (3) the patch is applied to the original code and executed via our evaluation infrastructure; and (4) the results are evaluated using specified metrics.

often showing qualitative signs of success even when they did not achieve final success. Nevertheless, even the best-performing agent succeeded only about one third of the time on average (33% success rate for OpenHands + Claude 4 Sonnet), leaving much headroom for progress.

While the current **REXBENCH** tasks pose substantial challenges for the agents tested, most extensions do not require major rewriting of the codebase and are not extremely challenging in terms of complexity (at least to a PhD-level domain expert). We thus consider the release of this specific set of tasks and the paper as a contribution about the broader framework for evaluating research extensions (and the opportunities it may bring), which will motivate the development of more challenging extensions covering broader scientific domains, inviting contributions from the community.

2 Related Work

Recent advancements in LLMs and agentic frameworks have motivated discussions about their applicability to scientific research. This includes using LLMs and LLM agents for research automation (Li et al., 2025a; Skarlinski et al., 2024; Jansen et al., 2025; Ziems et al., 2024; Choi, 2024; Boiko et al., 2023; Gottweis et al., 2025; Kitano, 2021; Gandhi et al., 2025; Mitchener et al., 2025) and benchmarking their ability to conduct research in the domains of social sciences, statistics, and natural sciences (Tian et al., 2024a; Chen et al., 2024; Laurent et al., 2024; Sharma et al., 2025). For ML research, current attempts use agents to automate all stages of the research process: from ideation (Si et al., 2024) to experiment design (Abramovich and Chechik, 2025; Liu et al., 2025; Zhao et al., 2025) and execution (Siegel et al., 2024; Xiang et al., 2025; Miao et al., 2025), to paper review and meta review (Du et al., 2024). There have also been early attempts to automate the full research pipeline (Lu et al., 2024; Kon et al., 2025; Miyai et al., 2025).

Another line of work benchmarks coding and software engineering skills. Specific skills targeted include improving iterative coding capabilities (Yang et al., 2025), resolving GitHub issues (Jimenez et al., 2024; Badertdinov et al., 2025; Deng et al., 2025), debugging LeetCode problems (Tian et al., 2024b), resolving configuration and dependency issues in research environment setups (Bogin et al., 2024), and solving tasks in a terminal environment (The Terminal-Bench Team, 2025). In a similar vein, other benchmarks assess more comprehensive ML problem-solving and code implementation skills. MLE-bench (Shern et al., 2024) and DSBench (Jing et al., 2024) design machine learning and data science tasks akin to Kaggle-style competitions; MLAgentBench (Huang et al., 2024) gathers classical ML tasks, including Kaggle challenges; DataSciBench evaluates data analysis and visualization skills with novel evaluation pipelines (Zhang et al., 2025); ML-Dev-Bench (Padigela et al., 2025) focuses on the full ML development workflow; and MLGym challenges agents' full pipeline research skills with tasks in domains such as computer vision and NLP (Nathani et al., 2025).

The most directly relevant efforts to ours are agentic frameworks and benchmarks that focus on ML problem-solving and software engineering capabilities in research settings. Curie (Kon et al., 2025) aims to evaluate the ability to plan and execute experiments; BLADE (Gu et al., 2024) is designed to automatically evaluate agents' approaches to open-ended data-driven research questions. ResearchCodeBench (Hua et al., 2025) tests LLMs' abilities to code ML ideas derived from recent research papers; Paper2Code (Seo et al., 2025) introduces a multi-agent LLM framework to trans-

late ML papers into codebases through a stage-wise design; PaperBench (Starace et al., 2025) evaluates research agents using a compilation of coding tasks targeting the replication of 20 ICML papers; and DeepCode (Li et al., 2025b) proposes agentic frameworks that targets long-context coding challenges, such as PaperBench tasks.

REXBENCH has a similar goal to PaperBench and, to some extent, Curie, in benchmarking ML and AI research code generation. However, a key distinction is that instead of evaluating replications (PaperBench) or very general questions that can often also be answered without running experiments (Curie), we focus on *novel research extensions*. Thus, **REXBENCH** is able to evaluate agent performance on previously unseen or unimplemented research hypotheses, which greatly alleviates data contamination concerns.

3 Benchmark Design

3.1 Research Extension Task

Task We define our research extension task as a code implementation problem, where the input consists of an existing research paper, an accompanying codebase, and an instruction that verbally describes an extension proposal and how this should be tested. An example of a simple extension is: "What would happen if the same experiment in paper X used an open-source model like Llama 3 70B instead of GPT-4o?" (see Appendix F for an actual example). Given this input, a system must produce as output edits to the input codebase that implements the extension proposal.

Desiderata The core aim of our benchmark is to *automatically* assess how well an agent can *autonomously* implement *realistic* research extensions. These goals are to some extent in conflict with each other. Realistic research extensions tend to be quite open-ended, which makes automatic assessment challenging or impossible. On the other hand, limiting tasks to ones that can be scored by simple automatic measures may constrain the task too much for it to be still realistic. We strike a balance between these two goals by using automatic tests that allow the agent to tackle the task through any means, as long as this leads to results comparable to the ones from our gold implementation. The task setting of requiring implementation on top of an existing codebase and evaluation through controlled execution environments (random seed, hardware, packages, etc.) serves to improve the reliability of the numeric output-based automatic evaluation. Nevertheless, each extension proposal included in the benchmark still cannot be too open-ended or exploratory, and therefore consist of specifically-scoped questions that can have well-defined numeric targets. To ensure that agents autonomously implement extensions, the granularity of our instructions are calibrated at a level that still requires the model to thoroughly analyze the codebase and form its own plan for the extension. Furthermore, at no point of the evaluation do humans provide additional supervision. Finally, one of the biggest challenges with LLM evaluation is data contamination. If solutions to any of the tasks are openly available on the web, LLMs that serve as the backbone for the agents may have been trained on the solutions (also noted as a possible issue in PaperBench: Starace et al. 2025), rendering it impossible to establish whether success stems from memorization or autonomously solving the task. We circumvent this problem by including only novel research extensions, either in terms of the idea itself or implementation. To the best of our knowledge, none of our extensions exist on top of the existing codebases publicly; we store all the gold extensions in private Bitbucket repositories.[1] Furthermore, our privately hosted evaluation infrastructure prevents models from accessing the evaluation scripts or reference solutions.

3.2 Benchmark Composition

Our benchmark consists of research extensions building upon papers and codebases primarily in the NLP and broader AI domains, taking into consideration the availability of expertise within the team as well as the availability/replicability of the code released. The full list of papers is in Table 1.[2] The specific extension proposals were selected to span various dimensions of change including changes to the model, dataset, algorithm, and evaluation. In addition to this consideration, we imposed the following constraints on the extension proposals for scientific rigor and feasibility of the experiments: (1) Important empirical trends from

[1] We use Bitbucket instead of GitHub since GitHub data has been used in the past to train LLMs and it is unclear whether this may also be true for some private repositories.

[2] Two of the tasks (COGS, Othello) involve implementing an extension proposal from another paper on top of the codebase of the original paper, where the implementation of the extension is either not publicly available or is not implemented as an edit of the original codebase. For these tasks, there are two relevant papers.

the original paper relevant to the extension proposal must replicate; (2) the gold implementation of the extension proposal must replicate (e.g., if the gold implementation requires making calls to a closed API-based model, this may not replicate in the future due to model deprecation); and (3) the estimated runtime of each gold implementation should be shorter than 12 hours on a single A100 GPU. The final dataset includes the extension instruction, target research papers in both .pdf and .md format (converted using `PyMuPDF4LLM` to accommodate agents that lack the ability to read .pdf files), and the original codebase.

3.3 Benchmark Construction Process

For each extension proposal, a domain expert (PhD student-level or above) first verified that the original codebase replicates the results of the associated paper on our virtual machines (details to follow). Then, they implemented the "gold" edits for the target extension and recorded the numerical outcomes, ensuring that the runtime does not exceed 12 hours. This implementation process and the outcomes were validated by at least one other author. Finally, the domain expert wrote the instruction that consists of a brief description of the original paper, the extension proposal, and how this proposal should be tested (see Appendix F for a full instruction for one of our tasks, WinoDict). The description of the "how" was deliberately high-level to meet the desideratum of evaluating a sufficiently autonomous capacity. Nevertheless, since the instructions should not be confusing or ambiguous, they were polished through multiple rounds of revisions by multiple authors to improve clarity. Importantly, if we foresaw degrees of implementation freedom that may introduce random variation, we controlled for this by specifying constraints (e.g., use an implementation of Pearson correlation function from the `scipy` package as opposed to implementing this from scratch). During this revision process we furthermore ensured that each extension was self-contained. No part of the gold edits required information external to the set of inputs provided to the system. As a part of the revisions for self-containment, we provided information such as specific model identifiers and explanations of necessary hyperparameters not in any README or the paper as a part of the instruction, and added version information for all of the packages (via an `environment.yml` file).

3.4 Evaluation Metrics

Our main metric is final success rate, which measures whether the outcome of executing the model-implemented code falls within the target range. We define two additional metrics for finer-grained analyses: execution success rate and file recall. We describe each metric below.

Final Success Rate Final success rate evaluates whether the model correctly implements the specified research extension. This evaluation either checks whether the final results exactly match the results of the reference implementation (if the run is fully deterministic) or it checks whether the results fall within a *gold range* that we obtained by running the gold implementation multiple times with different random seeds to account for output variability. In the latter case, a model solution is considered successful if its final execution outcome falls within this bound.

Execution Success Rate Execution success rate checks whether the generated code runs without errors in our evaluation environment. This metric evaluates the general well-formedness of the code and contextual understanding sufficient to avoid runtime errors.

File Recall File recall quantifies whether files edited in the gold solution were also edited by the model: $\text{File Recall} = |\text{Files}_{\text{agent}} \cap \text{Files}_{\text{gold}}| \,/\, |\text{Files}_{\text{gold}}|$. The limitation of this measure is the dependency on the gold solution. Technically, a solution could achieve zero file recall with perfect final success. E.g., if a model solution was exactly equivalent to gold but created new files with identical content instead of editing, and changed references appropriately in the repository, this would be the case. Still, we take human expert edits to reflect a reasonably efficient set of modifications.

3.5 Evaluation Infrastructure

Submission format Our metrics defined above require execution of agent generated code. We conduct this execution on a virtual machine to control for hardware specification and package dependencies. We host this infrastructure using our own resources to encourage community participation without resource concerns, and conduct evaluation asynchronously at a regular interval to update the leaderboard with the submissions we receive, similarly to Jimenez et al. (2024). The submissions are

received in the form of git patch files (as opposed to full edited repositories) to streamline the submission process. We also request agent log files to verify that the task was completed autonomously by an agent.

Infrastructure pipeline We host our evaluation infrastructure based on the OpenStack platform on an academic cloud computing service. For each patch file received, we execute the code in in a task-specific Apptainer container (Singularity, 2021) that has the original codebase and evaluation scripts pre-loaded and the environment set up. To control for random variation of the execution outcomes to the best of our effort, we (1) fix all random seeds in the codebase wherever possible, and (2) run the evaluations with exactly the same hardware configuration as our gold runs (see Appendix D, Table 3). Inside the container, we apply the patch file and execute the task. We limit the runtime to 12 hours, which is around twice the duration of the gold solution with the longest runtime among our extension tasks (see Table 3 for all estimated runtimes). Once task execution is complete or the attempt crashes, we extract result files and task execution logs. This setup ensures a fully containerized and task-level parallelizable evaluation infrastructure.

4 Experiments

4.1 Main Experiment

We follow steps shown in Figure 1 and evaluate twelve LLM agents, combining two agent frameworks with various LLM backbones (discussed below). We pass the full set of inputs for each task one by one to the agent to evaluate each task independently of each other. We run each task five times with the same agent model to account for agent random variation.

4.1.1 Baseline agent design

We used two different open-source agent frameworks (aider(aider AI, 2023) and OpenHands (Wang et al., 2025)) that we adapted for the task. aider and OpenHands both support multiple backbone LLMs. We evaluated `GPT-5`, `o1` and `o4-mini` (OpenAI), `Claude 3.7 Sonnet` and `Claude 4 Sonnet` (Anthropic), and the open-weight `DeepSeek-R1`. We discuss a few design decisions shared between our agents below. Note that this does not imply future submissions to our benchmark should be subject to the same design decisions.

Shared design considerations For better runtime controllability, we disabled Python code execution for all agents. Regarding the settings of the backbone LLMs, we set the temperature to 0.7 for Claude 4/3.7 Sonnet and DeepSeek-R1. For GPT-5, o1, and o4-mini, we used the default settings, as these models do not support custom temperature adjustment. We specified the reasoning effort as medium for all OpenAI models. As discussed in Section 3.5, our evaluation infrastructure requires git patch files. We created the patch files using a separate script after the agents had made changes to the codebase. We discuss individual implementation details in Appendix C.

4.2 Experiment with Additional Hints

We conduct an additional set of experiments where we provide different levels of hints to the agents. This experiment serves two purposes: (1) as a check that our tasks are possible to solve; (2) to gain a more fine-grained understanding of where the difficulties lie, if the agents do find the tasks difficult without hints. We design two levels of hints, where the first level of hints provides help with information localization, and the second level of hints provides a step-by-step implementation guidance. Information localization hints, for instance, help find specific locations of edits by directly naming a file to be edited ("You would need to edit `test_function()` in `src/testfile.py`"), help find necessary information ("Look at the README to find the descriptions of the hyperparameters"), or provide certain pieces of information that are part of the given input but nontrivial to find ("Use ID #1014 for the special token"). On the other hand, the second level of hints breaks down the gold solution into concrete implementation steps. Therefore, we expect the second level of hints to yield substantially higher success rates. In our experiments, hints are cumulative; when providing the second level of hints, the first level of hints is also provided.

4.3 Results

Main experiment Figure 2 shows our main results. Most agents struggle with the task, with the best performing agent (OpenHands + Claude 4 Sonnet) achieving a 33% average final success rate. All agents achieved nonzero execution success rates except for DeepSeek-R1, which failed completely. Claude 4 Sonnet again performed best, with a execution rate of 68% when combined with

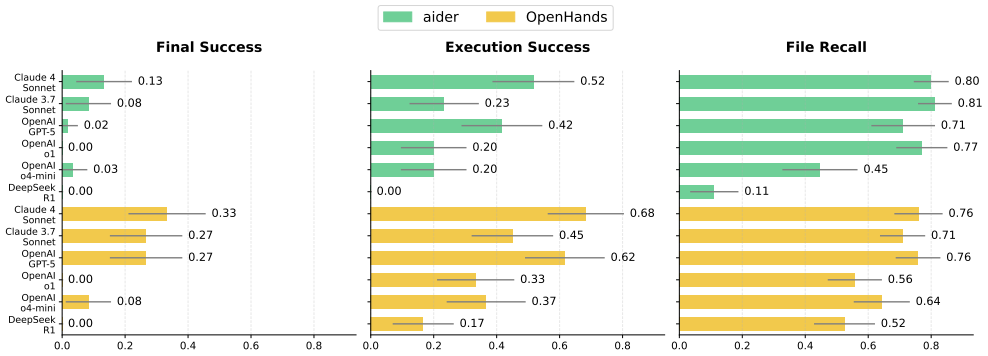

Figure 2: Agent performance on **REXBENCH**. The color coding indicates the agent framework and the y axis indicates the the backbone LLM. Results include five runs per task to account for agent random variation. Error bars show standard error of the mean of all runs per model computed using the closed form formula (2σ, no normality assumption).

OpenHands. The agents overall achieved high file recall, showing that they were able to locate core edit targets based on the instructions.

Additional hints Figure 3 (and Table 5 in Appendix H) shows the results of additional experiments with two different hint levels. Generally, hints improve the final success rate, but tend to help less when the default success rate was zero, suggesting there is a base level of competence required to make use of the hints provided. With the hints, we could boost the performance of two of the best agents, OpenHands + {Claude 4 Sonnet, GPT-5}, both achieving 43% final success rates.

4.4 Resource Consumption

Based on the final success rate, we plot the cost/-time vs. performance tradeoff (Figure 4), showing that aider + o4-mini, aider + Claude 4 Sonnet, OpenHands + Claude 3.7 Sonnet and OpenHands + Claude 4 Sonnet lie on the Pareto frontier for both cost and time. We provide the full time and cost estimates for agent runs in Appendix H, Table 7. In terms of token usage statistics, aider consistently used 2 turns due to its non-iterative design. OpenHands used the most tokens, especially with Claude 4 Sonnet, reaching up to 1.85M prompt tokens (almost 592 times more than aider). See Table 6 in Appendix H for token usage statistics by model and by hint levels.

5 Analysis and Discussion

5.1 Patterns of Error

We discuss notable error patterns, dividing them into explicit and implicit errors. We treat cases where the agent-generated code failed to execute as explicit errors, and cases where the execution succeeded but the experimental outcome did not match the numerical criteria as implicit.

Explicit errors Explicit errors were automatically identifiable from execution logs. The most common source of error was Python value errors (e.g., incorrect chat templates or invalid parameters). These errors were observed in all agents. Another common source of error was empty patch files due to the failure of the agent to modify any code. The majority of the empty patch file errors were from aider + {DeepSeek-R1, o4-mini}. We attribute this to the non-iterative nature of this agent framework: agents need to solve the entire extension task in one shot rather than breaking it down, often leading to incomplete or failed command executions during agent runs. Beyond these cases, most explicit errors were Python errors and they were mostly Python native errors rather than library-specific errors. Agents with Claude or GPT-5 as backbone led to fewer SyntaxErrors (in particular, OpenHands + {Claude 4/3.7 Sonnet, GPT-5} had no SyntaxErrors), whereas o1 produced SyntaxErrors frequently. There were also several cases of execution timeout, which occurs when the experiment runtime exceeds the limit of 12 hours we set (no gold solution required more than 6 hours). The full error distribution is shown in Figure 7 and Tables 8 and 9 in Appendix H.

Implicit errors Analysis of implicit errors (execution success but mismatch with gold outcome) involved greater manual effort because it required a holistic review of agent edits. Therefore, we focused our analysis on the top 2 agents (OpenHands + {Claude 4, GPT-5}). Overall, the agents' implicit

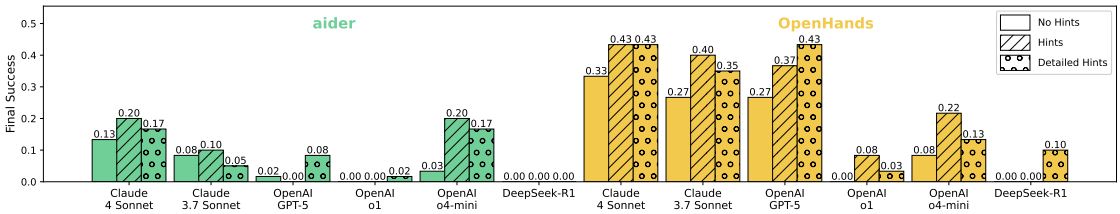

Figure 3: Final success rates for each agent-LLM combination and hint level.

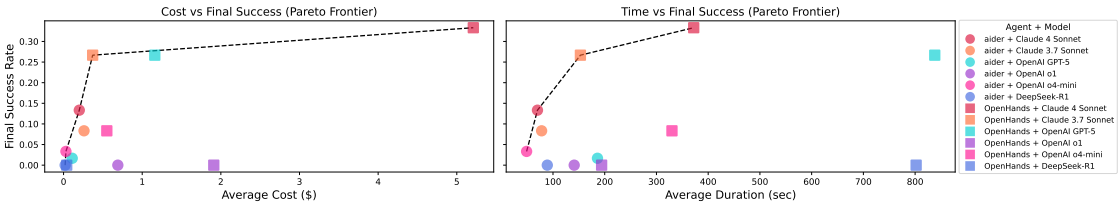

Figure 4: Cost effectiveness and time efficiency of coding agents on **REXBENCH**.

errors were categorizable into errors in implementation logic and errors in value (e.g., within-bounds index errors, incorrect hyperparameters or paths)—the ratio of logic vs. value errors was about 2:1. We also estimated the debugging difficulty from the manually identified sources of error, using the scale of easy (requires small local fix), medium (requires logical but local revisions), and hard (requires holistic revisions). The majority of the errors were easy to debug (16 easy, 4 medium, 4 hard). Many of the medium and hard implicit errors arose from the agent "over-editing" the code beyond the given instructions (e.g., adding extra (incorrect) exception handling or changing irrelevant flags/prompts). These unrequested edits often caused silent failures or subtle deviations from the gold implementation leading to markedly different results, making debugging harder. Task-specific examples are discussed in Appendix I.

5.1.1 Qualitative Observations

Implicit errors increase as model capacity increases, but are more difficult to analyze A high-level observation is a general pitfall associated with stronger models (for our task and coding tasks more generally): the cause of failure is difficult to identify. Better models produced more implicit errors (e.g., OpenHands + Claude 3.7: 6, OpenHands + Claude 4: 24), where the code successfully executes but the outcome is incorrect. In such cases, the reasons behind failure were not always easily traceable even for the experts who implemented the solutions. This highlights the need for rigorous tests if an agent were to be deployed

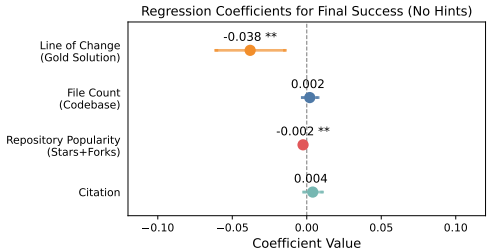

Figure 5: Regression coefficients with 95% confidence intervals for predictors of final success. (Regression model: `final_success` $\sim$ `line_change` + `file_count`+`repository_popularity`+`citation`+ (1 | `model`)). (*: $p < .05$, **: $p < .01$)

in practice. Plausible-looking implementations that execute can lead researchers to draw conclusions from faulty implementations, and over-reliance on coding agents may lead to a proliferation of incorrect results.

Overthinking is often an issue A prominent issue with weaker LLMs (Deepseek-R1, o1, o4-mini) was overthinking, where the thinking process was excessive both in terms of the number of output tokens and agent runtime, frequently leading to no actual output in terms of code generation. aider + DeepSeek-R1 was especially prone to this behavior, overthinking being one of the most prominent failure modes (close to one third of total failures). One possibility is that models' reasoning behavior somehow clashes with the reasoning/"thinking" loop of the agent framework, but this pattern appears weaker in Claude 4/3.7 Sonnet and GPT-5, which are themselves reasoning models.

Agents vary in their ability to make use of hints As noted in Section 4.3, providing additional hints did not always improve agents' success rates, nor did more hints necessarily yield greater gains. While the best agent (OpenHands + Claude 4) benefited from both levels of hints, many others showed no improvement. The best agents generally benefited more from the second level of hints, suggesting that a certain baseline competence or underlying model capacity may be required to leverage more detailed, human-written guidance. We observed idiosyncratic task-level variation as well; for instance, for the Othello task, OpenHands + Claude 3.7 Sonnet achieved 100% success rate with no hints and with the first level of hints, but 0% success rate when additionally given the second level of hints. Upon closer observation, the agent was unable to correctly use a helper function specified in the second level of hints, but with less detailed hints, it was able to correctly re-implement the helper function's logic. However, it was not the case that this particular hint was misleading, since the best performing agent was able to use this hint to achieve 100% success rate on this task. This can be interpreted as models varying in their ability to implement different equally plausible solutions, and the step-by-step guideline in the second level of hints specifying a different solution from the one that the model could implement easily.

5.2 What makes an extension difficult for agents?

We hypothesize four sources of difficulty that could contribute to agent failure: (1) implementation effort; (2) codebase size; (3) unfamiliarity with the codebase; and (4) unfamiliarity with the research topic. We operationalize them as: (1) lines of code change in our gold solution; (2) file counts of the original codebase; (3) GitHub stars + forks (repository popularity); and (4) Google Scholar citations of the research paper(s), respectively. We use these as predictors of final success in a mixed-effects model with model identity as a random effect. Figure 5 shows the regression coefficients. Lines of code changes has a significant negative effect ($\beta = -0.038$, $p < 0.01$) on final success, indicating that tasks with higher implementation effort are more difficult. Repository popularity had a significant effect but the effect size was negligible. Other factors were not statistically significant. We also compute cyclomatic complexity as a proxy for implementation difficulty but exclude it from the regression due to strong correlation with lines of code change (see Appendix E).

6 Conclusion

We presented **REXBENCH**, a benchmark evaluating the autonomous capacity of AI systems to implement hypothesis-driven research extensions in the domain of AI research. **REXBENCH** consists of realistic but well-scoped extension tasks motivated by existing research. To perform well, a system must be able to understand the expert-written extension instructions situated in specific research context, understand the structure and logic of the original codebase, and autonomously plan and implement the requested extension. Our tasks are by design robust to data contamination due to the extensions requiring novel implementations whose solutions are not available publicly. Experiments with various agent frameworks combined with competent backbone LLMs show that most systems struggle on our benchmark, with the best performing model (OpenHands + Claude 4 Sonnet) achieving a 33% success rate. Notably, agents with o1 or DeepSeek-R1 as backbone showed (close to) zero success rate. Nevertheless, closer analysis of the best-performing agents revealed promise: agents using the strongest backbone LLMs (Claude 4 Sonnet, GPT-5) achieved higher execution success rates than those using weaker models, with implementations often syntactically valid and logically on the right track. Overall, this observation, taken together with the large headroom, highlights the utility of **REXBENCH** for guiding future developments of research agents. We outline actionable modeling recommendations in Appendix B.

The future of REXBENCH Finally, as discussed in the introduction, we view the release of **REXBENCH** as a start to a larger community-driven effort. While our tasks were primarily in the NLP domain, we believe the format of the extension task and the evaluation framework are broadly applicable. We hope the setup draws community interest in research extensions for evaluating agents and we welcome community contributions for broader coverage of tasks.

Limitations

As discussed in Section 3.1, a benchmark task being realistic inherently conflicts with the ease of automatic evaluation. In particular, a task like research extension can be extremely open ended in

reality, even when constrained with a specific extension proposal and hypothesis. We opted for a middle ground where we do not enforce strong limitations on *how* a system may implement the target extension and condition final success on alignment of numerical outcomes. This necessitated a stronger control for sources of variation, which led us to write instructions as self-contained and unambiguous as possible. This setting is idealized in that they are much more informative and clearer than an actual task a human researcher may face, even in scenarios where the extension idea is provided to them (e.g., an advisor suggesting to a PhD student "How about we try X this time?"), missing out on the real difficulties lying in the initial trial-and-error concretization step.

Furthermore, while the three automatic metrics that we provide measure different aspects of success, additional process-level metrics such as landmark evaluation (Xu et al., 2024a; Bogin et al., 2024) would help alleviate the difficulty of post-hoc error analysis discussed in Section 5.1, especially for implicit errors, as well as reducing reward hacking or gamification of the benchmark. However, not all tasks are well-suited for such intermediate checks and additionally, their implementation requires substantial manual effort, which is why we do not provide such intermediate tests.

Additionally, the benchmark would benefit from a greater number of tasks. That being said, however, each task in our benchmark has high complexity as well as carrying signal; the current setup already clearly highlights performance differences between different agent setups and different backbone models. While the benchmark may currently not be able to reliably estimate differences between models with highly similar levels of abilities, our results demonstrate that it is effective at tracking overall progress as new models are being released (e.g., Claude 3.7 to 4).

Regarding broader societal impacts, the baseline agents we developed for this work did not reach the level of competence that we believe would translate into autonomous research extension capacities in the real world. Still, our benchmark may contribute to developing such systems in the future, which may have positive impacts such as contributing to better replicability and faster iterations of empirical hypothesis verification. On the other hand, given the difficulty of debugging errors, deployment of such systems without rigorous verification measures faces the danger of leading researchers to draw conclusions from faulty implementations and of the erosion of trust in published results.

Ethical considerations

In this work, we showed that current LLM-based agents cannot reliably produce code for AI research without additional human supervision. We based this argument on the low final success rate of all evaluated agents, as well as the danger of the increasing trend of implicit errors as model capacity improves. Given the rapid progress of AI research and model development, it is a likely possibility that new agents would perform significantly better on this benchmark in the near future. The biggest risk we therefore foresee is that good performance on this benchmark is seen as a sufficient condition for reliable agents rather than a necessary one. While we consider the benchmark to be well-suited for measuring progress in the development of future agents, good performance should NOT be seen as sufficient evidence for an agent being able to autonomously produce reliable research code.

Executing machine-written code always bears safety risks and providing AI agents with too much freedom for exploration may enable them to cause harm. To mitigate this risk, we narrowly scoped the implementation tasks in our experiments fully based on human-generated hypotheses and instructions. Furthermore, any machine-written code was executed in a containerized environment without internet access. We recommend similar setups for the execution of any code that is output by AI agents.

References

Talor Abramovich and Gal Chechik. 2025. Ablationbench: Evaluating automated planning of ablations in empirical ai research. *arXiv preprint arXiv:2507.08038*.

aider AI. 2023. aider: Ai pair programming in your terminal. https://github.com/Aider-AI/aider. Accessed: 2025-05-12.

Anthropic. 2024. Claude 3.7 sonnet system card. https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf. Accessed: 2025-05-14.

Anthropic. 2025. Claude 4 sonnet system card. https://www-cdn.anthropic.com/6d8a8055020700718b0c49369f60816ba2a7c285.pdf. Accessed: 2025-09-22.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. 2025. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. 2024. SUPER: Evaluating agents on setting up and executing tasks from research repositories. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12622–12645, Miami, Florida, USA. Association for Computational Linguistics.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, and 1 others. 2024. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery, october 2024. *arXiv:2410.05080*.

Jonathan H. Choi. 2024. How to use large language models for empirical legal research. *Journal of Institutional and Theoretical Economics (JITE)*, 180(2):214–233.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 619–634, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, and 1 others. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*.

Jiangshu Du, Yibo Wang, Wenting Zhao, Zhongfen Deng, Shuaiqi Liu, Renze Lou, Henry Peng Zou, Pranav Narayanan Venkit, Nan Zhang, Mukund Srinath, Haoran Ranran Zhang, Vipul Gupta, Yinghui Li, Tao Li, Fei Wang, Qin Liu, Tianlin Liu, Pengzhi Gao, Congying Xia, and 21 others. 2024. LLMs assist NLP researchers: Critique paper (meta-)reviewing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5081–5099, Miami, Florida, USA. Association for Computational Linguistics.

Julian Martin Eisenschlos, Jeremy R. Cole, Fangyu Liu, and William W. Cohen. 2023. WinoDict: Probing language models for in-context word acquisition. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 94–102, Dubrovnik, Croatia. Association for Computational Linguistics.

Kanishk Gandhi, Michael Y Li, Lyle Goodyear, Louise Li, Aditi Bhaskar, Mohammed Zaman, and Noah D Goodman. 2025. Boxinggym: Benchmarking progress in automated experimental design and model discovery. *arXiv:2501.01540*.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, and 1 others. 2025. Towards an ai co-scientist. *arXiv:2502.18864*.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, Yikun Zhang, Tianmai M. Zhang, Lanyi Zhu, Mike A Merrill, Jeffrey Heer, and Tim Althoff. 2024. BLADE: Benchmarking language model agents for data-driven science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13936–13971, Miami, Florida, USA. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv:2501.12948*.

John Hewitt, Nelson F. Liu, Percy Liang, and Christopher D. Manning. 2024. Instruction following without instruction tuning. *arXiv:2409.14254*.

Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. 2025. Researchcodebench: Benchmarking llms on implementing novel machine learning research code. *arXiv preprint arXiv:2506.02314*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. Mlagentbench: evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, pages 20271–20309.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. OpenAI o1 system card. *arXiv:2412.16720*.

Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Daniel S. Weld, and Peter Clark. 2025. Codescientist: End-to-end semi-automated scientific discovery with code-based experimentation. *arXiv:2503.22708*.

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

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. Dsbench: How far are data science agents to becoming data science experts? *arXiv:2409.07703*.

Julie Kallini, Isabel Papadimitriou, Richard Futrell, Kyle Mahowald, and Christopher Potts. 2024. Mission: Impossible language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14691–14714, Bangkok, Thailand. Association for Computational Linguistics.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Najoung Kim, Sebastian Schuster, and Shubham Toshniwal. 2024. Code pretraining improves entity tracking abilities of language models. *arXiv:2405.21068*.

Hiroaki Kitano. 2021. Nobel turing challenge: creating the engine for scientific discovery. *NPJ systems biology and applications*, 7(1):29.

Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. 2025. Curie: Toward rigorous and automated scientific experimentation with ai agents. *arXiv:2502.16069*.

Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D. White, and Samuel G. Rodriques. 2024. Lab-bench: Measuring capabilities of language models for biology research. *arXiv:2407.10362*.

Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Pilsung Kang, and Najoung Kim. 2025. Checkeval: A reliable LLM-as-a-judge framework for evaluating text generation using checklists. *arXiv:2403.18771*.

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*.

Sihang Li, Jin Huang, Jiaxi Zhuang, Yaorui Shi, Xiaochen Cai, Mingjun Xu, Xiang Wang, Linfeng Zhang, Guolin Ke, and Hengxing Cai. 2025a. ScilitLLM: How to adapt LLMs for scientific literature understanding. In *The Thirteenth International Conference on Learning Representations*.

Zongwei Li, Zhonghang Li, Zirui Guo, Xubin Ren, and Chao Huang. 2025b. Deepcode: Open agentic coding. *arXiv preprint arXiv:2512.07921*.

Yaowenqi Liu, BingXu Meng, Rui Pan, Yuxing Liu, Jerry Huang, Jiaxuan You, and Tong Zhang. 2025. Guide: Towards scalable advising for research ideas. *arXiv preprint arXiv:2507.08870*.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv:2408.06292*.

Ziming Luo, Zonglin Yang, Zexin Xu, Wei Yang, and Xinya Du. 2025. Llm4sr: A survey on large language models for scientific research. *arXiv:2501.04306*.

Chunyu Miao, Henry Peng Zou, Yangning Li, Yankai Chen, Yibo Wang, Fangxin Wang, Yifan Li, Wooseong Yang, Bowei He, Xinni Zhang, and 1 others. 2025. Recode-h: A benchmark for research code development with interactive human feedback. *arXiv preprint arXiv:2510.06186*.

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, Eric C Landsness, Daniel L Barabasi, Siddharth Narayanan, Nicky Evans, and 1 others. 2025. Kosmos: An ai scientist for autonomous discovery. *arXiv preprint arXiv:2511.02824*.

Atsuyuki Miyai, Mashiro Toyooka, Takashi Otonari, Zaiying Zhao, and Kiyoharu Aizawa. 2025. Jr. ai scientist and its risk report: Autonomous scientific exploration from a baseline paper. *arXiv preprint arXiv:2511.04583*.

Neel Nanda, Andrew Lee, and Martin Wattenberg. 2023. Emergent linear representations in world models of self-supervised sequence models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 16–30, Singapore. Association for Computational Linguistics.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, and 1 others. 2025. Mlgym: A new framework and benchmark for advancing ai research agents. 2025. *URL https://doi.org/10.48550/arXiv*.

OpenAI. 2025. Gpt-5 system card. `https://cdn.openai.com/gpt-5-system-card.pdf`. Accessed: 2025-09-22.

Harshith Padigela, Chintan Shah, and Dinkar Juyal. 2025. Ml-dev-bench: Comparative analysis of ai agents on ml development workflows. *arXiv:2502.00964*.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2025. Paper2code: Automating code generation from scientific papers in machine learning. *arXiv:2504.17192*.

Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, and 1 others. 2025. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents. *arXiv preprint arXiv:2511.07685*.

Chan Jun Shern, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2024. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv:2410.07095*.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. 2024. Can LLMs generate novel research ideas? a large-scale human study with 100+ NLP researchers. *arXiv:2409.04109*.

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

Singularity. 2021. Singularity.

Michael D. Skarlinski, Sam Cox, Jon M Laurent, James D. Braza, Michaela Hinks, Michael J. Hammerling, Manvitha Ponnapati, Samuel G. Rodriques, and Andrew D. White. 2024. Language agents achieve superhuman synthesis of scientific knowledge. *arXiv:2409.13740*.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, and 1 others. 2025. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv:2504.01848*.

Zilu Tang, Mayank Agarwal, Alexander Shypula, Bailin Wang, Derry Wijaya, Jie Chen, and Yoon Kim. 2023. Explain-then-translate: an analysis on improving program translation with self-generated explanations. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1741–1788, Singapore. Association for Computational Linguistics.

The Terminal-Bench Team. 2025. Terminal-bench: A benchmark for ai agents in terminal environments.

Minyang Tian, Luyu Gao, Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, HAO TONG, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Shengzhu Yin, and 10 others. 2024a. Scicode: A research coding benchmark curated by scientists. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Hui Haotian, Liu Weichuan, Zhiyuan Liu, and Maosong Sun. 2024b. DebugBench: Evaluating debugging capability of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4173–4198, Bangkok, Thailand. Association for Computational Linguistics.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Leon Weber-Genzel, Siyao Peng, Marie-Catherine De Marneffe, and Barbara Plank. 2024. VariErr NLI: Separating annotation error from human label variation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2256–2269, Bangkok, Thailand. Association for Computational Linguistics.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1819–1862, Mexico City, Mexico. Association for Computational Linguistics.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. 2025. Scireplicate-bench: Benchmarking llms in agent-driven algorithmic reproduction from research papers. *arXiv:2504.00255*.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, and 2 others. 2024a. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *arXiv:2412.14161*.

Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, Jian-Guang Lou, and Shuai Ma. 2024b. Re-reading improves reasoning in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15549–15575, Miami, Florida, USA. Association for Computational Linguistics.

John Yang, Kilian Lieret, Joyce Yang, Carlos E Jimenez, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025. Codeclash: Benchmarking goal-oriented software engineering. *arXiv preprint arXiv:2511.00839*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving

with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025. Datascibench: An LLM agent benchmark for data science. *arXiv:2502.13897*.

Yilun Zhao, Weiyuan Chen, Zhijian Xu, Manasi Patwardhan, Chengye Wang, Yixin Liu, Lovekesh Vig, and Arman Cohan. 2025. Abgen: Evaluating large language models in ablation study design and evaluation for scientific research. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12479–12491.

Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. 2024. Can large language models transform computational social science? *Computational Linguistics*, 50(1):237–291.

A List of papers for RExBench extension

Table 1 shows the papers forming the basis of the RExBench extensions.

B Actionable recommendations

Based on the analyses of the agents tested in this work, we put forward several actionable recommendations for the future.

Short-horizon recommendations:

- **Incorporate iterative design**: Our findings show that iterative design is critical for success on our tasks: aider (a single-turn framework) showed weaker performance in general, and many success scenarios for multi-turn agents could be attributed to effective use of the previous turns' output. For instance, in the CheckEval task, OpenHands + GPT-5 used one turn to inspect a file's structure with bash before writing code in the next.
- **Support scratchpads**: Agents frequently failed on the basis of small errors such as path mis-specification. Such errors could be easily caught if agents can make use of a "scratchpad" where small code snippets can be executed.
- **Support "repair" mechanism**: Agents should incorporate a mechanism to repair a step in their action trajectory, for instance by reverting the changes made in the step and re-initiating the LLM call. One use case of this would be detecting and repairing overthinking in the LLM output, which was a prominent failure mode in several agents, especially with DeepSeek-R1 as backbone, that resulted in no code edits.

Longer-horizon recommendations:

- **More stringent verification**: One of the most concerning observations from our analysis is the increasing trend of implicit errors as the capacity of the backbone LLM grows. Under a benchmarking setup, numeric mismatches of the outcome to the gold solution easily indicates failure, but in real deployment scenarios, there exist no gold solutions. This indicates a need for more stringent verification processes, ideally by agent design rather than relying on manual verification from the end users.
- **Prevent over-editing**: A prominent failure mode of the strongest agents was "over-editing", where agents make unrequested modifications that often lead to implicit errors. Our findings show that simply instructing an agent to "keep everything else not specified constant" (see Appendix F) is insufficient. A general improvement in hallucination reduction and instruction-following would help, but for research coding where fine-grained controls of experimental details is critical, a more targeted solution for over-editing may be beneficial.
- **Improve handling of long contexts**: Our analysis shows that the most important factor to agent failure is the size of the required edits. Given that the maximum lines of change in the gold solutions in our benchmark is not huge (in the magnitude of hundreds), there is a need for future agents to handle long contexts better, both within and across file boundaries.

C Detailed Agent Configurations

Table 2 provides an overview of what kind of abilities each agent has.

aider aider is an open-source agent framework. We implemented our most basic agent based on aider, using the "diff" edit format where the LLM specifies file changes as search/replace blocks. We allowed up to 5 retries to handle API-side overload errors. Since aider lacks built-in file search capabilities, we added a preliminary stage where the LLM is given the codebase's directory tree along with the

Identifier	Extension Type	Source paper	Venue
CheckEval	Evaluation	*CheckEval: A reliable LLM-as-a-Judge framework for evaluating text generation using checklists (Lee et al., 2025)*	EMNLP 2025
COGS	Model	*COGS: A Compositional Generalization Challenge Based on Semantic Interpretation (Kim and Linzen, 2020)*	EMNLP 2020
		The Devil is in the Detail: Simple Tricks Improve Systematic Generalization of Transformers (Csordás et al., 2021)	EMNLP 2021
Entity Tracking	Model	*Code Pretraining Improves Entity Tracking Abilities of Language Models (Kim et al., 2024)*	Preprint
Explain then Translate	Algorithm	*Explain-then-Translate: An Analysis on Improving Program Translation with Self-generated Explanations (Tang et al., 2023)*	EMNLP Findings 2023
Instruction Tuning	Model	*Instruction Following without Instruction Tuning (Hewitt et al., 2024)*	Preprint
Mission Impossible	Data/Evaluation	*Mission: Impossible Language Models (Kallini et al., 2024)*	ACL 2024
Othello	Data/Evaluation	*Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task (Li et al., 2023)*	ICLR 2023
		Emergent Linear Representations in World Models of Self-Supervised Sequence Models (Nanda et al., 2023)	BlackboxNLP 2023
Reasoning or Reciting	Model	*Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks (Wu et al., 2024)*	NAACL 2024
Re-reading	Algorithm	*Re-Reading Improves Reasoning in Large Language Models (Xu et al., 2024b)*	EMNLP 2024
Tree of Thoughts	Algorithm	*Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Yao et al., 2023)*	NeurIPS 2023
VariErr-NLI	Model/Data	*VARIERR NLI: Separating Annotation Error from Human Label Variation (Weber-Genzel et al., 2024)*	ACL 2024
WinoDict	Data/Evaluation	*WinoDict: Probing language models for in-context word acquisition (Eisenschlos et al., 2023)*	EACL 2023

Table 1: List of papers that form the bases for extensions in **REXBENCH**.

Component	aider	OpenHands
Repo navigation	×	✓
Tool use	×	✓
Bash execution	×	✓
Python execution	×	×

Table 2: Agent components

task instruction to identify files requiring modification. Unlike OpenHands, our aider implementation does not use bash execution or tools.

OpenHands OpenHands is an open-source agent framework that uses an LLM to control a range of pre-defined tools for understanding and modifying codebases. We modified the system prompt and the agent to disable execution of Python code. The agent was allowed to execute bash commands such as `grep` and `cat`, browse the web, load PDFs in a browser (if compatible with the backbone LLM), and edit files. We prompted this agent with the same one-line instruction. We evaluated this agent in "headless" mode in which the agent executes the task without any user input until the LLM signals task completion to the agent, or the agent detects a loop or reaches a maximum number of steps (250). We applied postprocessing to absolute filepaths to make them compatible with the virtual machine evaluation environment, since the OpenHands agent is run inside its own Docker container.

D Detailed Experimental Setup

Task	Instance Type	Runtime Duration (Gold Solution)
CheckEval	CPU	1m
COGS	K80	5h
Entity Tracking	A100	2h
Explain then Translate	CPU	<1m
Instruction Tuning	A100	5h
Mission Impossible	A100	4h
Othello	K80	1h
Reasoning or Reciting	A100	6h
Re-reading	A100	30m
Tree of Thoughts	A100	20m
VariErr-NLI	A100	10m
WinoDict	A100	30m

Table 3: Resource requirements for each task.

Table 3 shows the details about the execution environment for each task.

E Solution Complexity

Table 4 shows the total net change (delta) in cyclomatic complexity between the original repository

Task	Delta Cyclomatic Complexity
CheckEval	93.0
COGS	3.0
Entity Tracking	1.0
Explain then Translate	12.0
Instruction Tuning	29.0
Mission Impossible	19.0
Othello	8.0
Reasoning or Reciting	16.0
Re-reading	2.0
Tree of Thoughts	18.0
VariErr-NLI	25.0
WinoDict	28.0

Table 4: Total net change (delta) in cyclomatic complexity between the original repository state and the gold implementation, computed over the targeted (modified) files.

state and the gold implementation, computed only over the files modified in the gold implementation. Cyclomatic complexity was measured using radon for Python and shellmetrics for bash. Delta cyclomatic complexity is strongly correlated with lines of code changed (Pearson $r = 0.845$).

F An Example Task Instruction (Extension of WinoDict)

G Tool use/action distribution

OpenHands agents interact with external tools during execution, and we analyze how their tool usage varies across different LLMs. Claude 4 Sonnet and OpenAI GPT-5 showed the highest overall tool usage. File operations (`str_replace_editor`) and bash commands (`execute_bash`) were the most frequently used tools across all models (see Figure 6) but occasionally the agent did also perform web searches or use a browser to render the paper PDF.

H Detailed Experimental Results

Table 5 shows the detailed results for all metrics and each agent-LLM combination across all three hint levels.

Table 6 shows the number of turns as well as the number of input and output tokens, averaged across the three runs for each agent.

Table 7 shows the costs and duration for running each agent on a single task on average, as well as the total cost and total durations, based on the main experiment only (providing no hints). Including preliminary and failed runs not reported in the main paper, we estimate that the total compute required for the full project was approximately 4–5x the reported amount.

Table 8 and Table 9 shows the detailed breakdown of errors for each agent and LLM combination.

Tables 10 to 21 show the detailed breakdown of task specific performance for each agent and LLM combination.

I Additional qualitative observations

Some agent edits have no practical effect Although stronger agents more often write executable code, sometimes the actual implementation has no effect on the output. For example, in the Mission Impossible task, both OpenHands + {Claude 4 Sonnet, GPT-5} incorrectly used the `ParentedTree` class in the `nltk` library. While this code raised a `ValueError`, the agent's implementation used a `try-except` block, returning the parse tree from the original paper as a fallback value, meaning the script still executed. In another instance, OpenHands + GPT-5 incorrectly tried to access the content returned by a method in the `radon` library. The code logic meant that if no function was found using this method, a default value of 0.0 was returned. This in effect meant that the final numerical results were identical to the original paper's experiments. These observations reaffirm the importance of rigorous verification before deploying these systems in the real world.

Detailed observations on over-editing As mentioned in Section 5.1, despite the constraints specified in our task instructions for the agents, we occassionally observed agents making unnecessary additional code edits. For example, for the Re-reading task, OpenHands + Claude Sonnet 4 unnecessarily modified an additional metadata field in one of the `.yml` files, which is used as part of the input in one of the experimental settings. Similarly, in the VariErr-NLI task OpenHands + GPT-5 unnecessarily modified an output file path required for obtaining the final scores, resulting in the evaluation scripts being unable to access the results of the agent's implementation. Given that scientific work relies on rigor and reproducibility, deviations like these from the specified instructions are problematic. This highlights the need to design agents which conform exactly to the requirements given, without introducing additional unrequested changes.

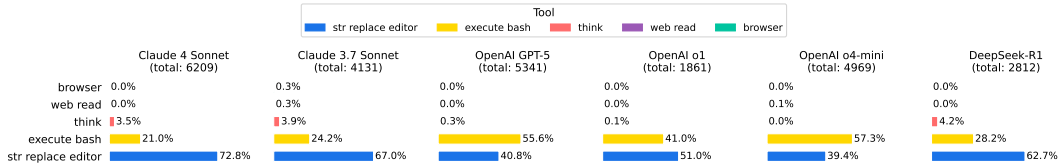

Figure 6: Tool usage distribution across OpenHands agent implementations. Percentages indicate the frequency of each tool type, while the total usage count is shown in each column header.

J License Information

The codebase portion of **REXBENCH** is constructed from public repositories—details of the licenses for each task are provided in Table 22. When the codebase did not contain any license information, we reached out to the authors for more information and used their suggestion (one response still pending at the time of writing, but we make an educated guess that the repository will be associated with a permissive license given that the paper was written by authors with primarily academic affiliations, and from the fact that the public availability of their codebase is mentioned in the paper). We release our data and code under a dual license (MIT and Apache 2.0), given the mixed license of the repositories included in the full benchmark suite.

Agent	Model	Hint Level	Final Success	Execution Success	File Recall
aider	Claude 4 Sonnet	No hints	0.13	0.52	0.80
		Hints	0.20	0.47	0.78
		Detailed Hints	0.17	0.42	0.83
	Claude 3.7 Sonnet	No hints	0.08	0.23	0.81
		Hints	0.10	0.18	0.79
		Detailed Hints	0.05	0.20	0.81
	OpenAI GPT-5	No hints	0.02	0.42	0.71
		Hints	0.00	0.38	0.74
		Detailed Hints	0.08	0.33	0.74
	OpenAI o1	No hints	0.00	0.20	0.77
		Hints	0.00	0.28	0.78
		Detailed Hints	0.02	0.35	0.77
	OpenAI o4-mini	No hints	0.03	0.20	0.45
		Hints	0.20	0.27	0.47
		Detailed Hints	0.17	0.33	0.51
	DeepSeek-R1	No hints	0.00	0.00	0.11
		Hints	0.00	0.00	0.05
		Detailed Hints	0.00	0.00	0.07
OpenHands	Claude 4 Sonnet	No hints	0.33	0.68	0.76
		Hints	0.43	0.68	0.83
		Detailed Hints	0.43	0.78	0.89
	Claude 3.7 Sonnet	No hints	0.27	0.45	0.71
		Hints	0.40	0.53	0.81
		Detailed Hints	0.35	0.52	0.87
	OpenAI GPT-5	No hints	0.27	0.62	0.76
		Hints	0.37	0.63	0.85
		Detailed Hints	0.43	0.70	0.84
	OpenAI o1	No hints	0.00	0.33	0.56
		Hints	0.08	0.33	0.64
		Detailed Hints	0.03	0.38	0.73
	OpenAI o4-mini	No hints	0.08	0.37	0.64
		Hints	0.22	0.37	0.73
		Detailed Hints	0.13	0.48	0.71
	DeepSeek-R1	No hints	0.00	0.17	0.52
		Hints	0.00	0.13	0.60
		Detailed Hints	0.10	0.22	0.68

Table 5: Detailed performance on **REXBENCH**, evaluated across three hint levels. Results are averaged across three runs.

WinoDict Task Instruction

Problem Description

Background. The paper *WinoDict: Probing Language Models for In-Context Word Acquisition* (Eisenschlos et al., 2023) studies the ability of large language models to acquire novel word meanings during inference. The authors rewrite Winograd-style coreference resolution problems by replacing a key concept word with a synthetic but plausible English word, and appending a definition of the new concept as a suffix.

Building on this work, we consider a learning setting in which the surface form of the learned word coincides with an existing English word. In this setting, the model must override or suppress the existing meaning of the word using only the provided definition. We hypothesize that overriding existing words is more difficult than learning entirely novel forms, and that the frequency of the original word may further modulate this effect.

The original paper is provided in the repository in both PDF format (`eisenschlos_et_al_2023.pdf`) and Markdown format (`eisenschlos_et_al_2023.md`) for reference.

Extension to Be Implemented. Your task is to modify the provided codebase to generate new WinoDict datasets by replacing the target word being learned with an existing English word. The newly generated datasets should be stored under the directory `./data`.

Word replacement must respect the part-of-speech (POS) tag of the original word. We consider four POS categories: nouns, verbs, adjectives, and adverbs. To examine the effect of word frequency, candidate replacement words should be sampled from three frequency-based groups:

- **Top group.**
  - Verbs, nouns, adverbs: top 20% most frequent words
  - Adjectives: top 35% most frequent words
- **Bottom group.**
  - Verbs, nouns, adverbs: bottom 20% least frequent words
  - Adjectives: bottom 35% least frequent words
- **All group.**
  - Verbs, nouns, adjectives, adverbs: all available words with no frequency-based filtering

Frequency information is provided via four files under the directory `./words/`: `1_all_rank_noun.txt`, `2_all_rank_verb.txt`, `3_all_rank_adjective.txt`, and `4_all_rank_adverb.txt`. Each file lists words in descending order of frequency according to the British National Corpus. You should construct candidate lists from these files and sample replacement words accordingly.

Replacement words must be inflected to match the morphological properties of the original word. For example, if the original word is a past-tense verb, the replacement word must also be inflected in the past tense. Please use `spaCy` together with the `lemminflect` module to perform the necessary inflection.

Using the newly generated datasets, run experiments on the WinoDict–Winograd dataset under a 5-shot setting. The model checkpoint can be found at `/stage/hf_cache/gemma-2-9b`, although it may not be visible in the repository. We only consider the setting in which definitions are appended as suffixes, corresponding to the `last_def` template in the codebase.

Save the experimental results as three JSON files under `./results/`, named `res_top.json`, `res_bottom.json`, and `res_all.json`, corresponding to the three frequency groups.

Implement a single script named `run_final.sh` in the root directory of the repository. This script should run end-to-end without any command-line arguments and should handle both dataset generation and experimental execution.

Please keep all aspects of the original repository unchanged unless explicitly specified above. The environment is already configured; do not install additional dependencies or download extra datasets or models. Refer to `environment.yml` for the list of available libraries and versions.

Evaluation. We will evaluate the extension by executing the provided `run_final.sh` script and inspecting the three output JSON files listed above.

Agent	Model	Hints Level	Total Turns (Avg.)	Prompt Tokens (Avg.)	Output Tokens (Avg.)
aider	Claude 4 Sonnet	No hints	2.00	2941.70	4048.60
		Hints	2.00	3028.20	4275.00
		Detailed Hints	2.00	3412.40	4187.10
	Claude 3.7 Sonnet	No hints	2.00	2941.80	4669.20
		Hints	2.00	3029.10	4021.70
		Detailed Hints	2.00	3413.00	3950.10
	OpenAI GPT-5	No hints	2.00	2881.20	8650.00
		Hints	2.00	2976.60	8721.40
		Detailed Hints	2.00	3360.50	9650.00
	OpenAI o1	No hints	2.00	2964.70	5536.30
		Hints	2.00	3061.80	6086.70
		Detailed Hints	2.00	3447.50	6088.30
	OpenAI o4-mini	No hints	2.00	2910.40	3944.60
		Hints	2.00	3002.40	3860.30
		Detailed Hints	2.00	3389.30	4648.10
	DeepSeek-R1	No hints	2.00	2966.10	3732.20
		Hints	2.00	3002.60	3850.90
		Detailed Hints	2.00	3446.20	3570.60
OpenHands	Claude 4 Sonnet	No hints	99.33	1,847,555.70	9872.00
		Hints	97.12	1,822,132.70	9791.90
		Detailed Hints	95.33	1,390,346.00	9814.20
	Claude 3.7 Sonnet	No hints	44.90	476,605.30	6932.30
		Hints	45.18	488,315.90	6850.60
		Detailed Hints	51.68	561,399.10	7344.30
	OpenAI GPT-5	No hints	69.62	871,624.30	37,786.20
		Hints	71.63	1,010,020.60	40,139.40
		Detailed Hints	67.30	947,675.20	34,107.50
	OpenAI o1	No hints	26.12	164,538.00	18,007.30
		Hints	22.73	131,386.80	12,660.40
		Detailed Hints	17.30	92,010.20	10,151.30
	OpenAI o4-mini	No hints	59.53	617,804.50	28,767.90
		Hints	58.95	592,439.80	27,485.30
		Detailed Hints	52.60	520,614.00	24,594.10
	DeepSeek-R1	No hints	36.10	270,543.70	18,085.10
		Hints	35.93	220,510.30	17,681.90
		Detailed Hints	35.72	211,106.10	16,938.80

Table 6: Token usage statistics across agents and models.

Agent	Model	Avg. Cost ($)	Avg. Duration	Total Cost ($)	Total Duration
aider	Claude 4 Sonnet	0.20	1m 10s	35.13	3h 28m 38s
	Claude 3.7 Sonnet	0.26	1m 18s	45.99	3h 54m 57s
	OpenAI GPT-5	0.11	3m 6s	19.47	9h 16m 45s
	OpenAI o1	0.69	2m 21s	124.69	7h 1m 34s
	OpenAI o4-mini	0.03	49s	5.45	2h 27m 25s
	DeepSeek-R1	0.02	1m 29s	3.94	4h 22m 7s
OpenHands	Claude 4 Sonnet	5.21	6m 12s	937.34	18h 38m 42s
	Claude 3.7 Sonnet	0.37	2m 33s	67.20	7h 41m 29s
	OpenAI GPT-5	1.16	13m 58s	208.07	41h 56m 30s
	OpenAI o1	1.91	3m 14s	343.44	9h 42m 56s
	OpenAI o4-mini	0.55	5m 30s	98.97	16h 31m 49s
	DeepSeek-R1	0.04	13m 22s	7.99	40h 8m 16s

Table 7: Cost and duration statistics across agents and models (main experiment).

Error Type	Aider					
	Claude 4 Sonnet	Claude 3.7 Sonnet	OpenAI GPT-5	OpenAI o1	OpenAI o4-mini	DeepSeek R1
Python Errors						
AssertionError	2	0	5	0	2	0
AttributeError	6	11	6	9	3	0
FileNotFoundError	21	73	11	19	7	3
ImportError	1	6	3	10	0	0
IndentationError	0	0	0	0	0	0
IndexError	0	0	3	2	0	0
IsADirectoryError	0	0	0	0	0	0
KeyError	1	5	2	3	12	2
LookupError	0	0	1	0	0	0
ModuleNotFoundError	2	0	7	4	0	0
NameError	1	1	1	1	0	1
NotImplementedError	0	0	0	0	0	0
OSError	0	0	0	0	0	0
RuntimeError	0	0	2	2	0	0
SyntaxError	1	0	0	10	6	0
TypeError	5	4	7	3	11	6
UnboundLocalError	2	0	0	2	0	0
ValueError	39	37	35	18	10	7
EOFError	0	1	0	0	0	0
Python Library Errors						
DatasetNotFoundError	1	0	0	0	0	0
NotFoundError	2	0	1	2	0	0
OutOfMemory	0	0	0	0	0	0
ArgumentError	0	0	0	0	0	0
ScannerError	0	0	0	0	0	0
Other Python Errors						
ConstructorError	2	0	0	0	0	0
JSONDecodeError	3	0	5	0	0	0
HFValidationError	0	0	0	0	0	0
ParserError	0	0	0	0	0	0
Bash Errors						
cannot create directory	0	0	0	1	0	0
empty patch	5	0	13	0	69	111
empty or missing	0	0	5	7	7	41
unable to write file	0	0	0	0	0	5
Permission denied	0	0	0	3	0	0
syntax error	0	0	0	0	0	0
cannot access	0	0	0	0	0	0
patch failed	1	0	0	0	0	0
Execution Timeout	0	2	0	12	1	2

Table 8: Breakdown of error counts for Aider.

Error Type	OpenHands					
	Claude 4 Sonnet	Claude 3.7 Sonnet	OpenAI GPT-5	OpenAI o1	OpenAI o4-mini	DeepSeek R1
Python Errors						
AssertionError	0	0	1	3	3	0
AttributeError	0	10	1	3	3	8
FileNotFoundError	6	11	1	4	2	7
ImportError	0	0	0	0	1	4
IndentationError	0	0	4	18	14	19
IndexError	6	2	5	1	0	2
IsADirectoryError	0	0	0	0	0	2
KeyError	1	6	1	2	2	14
LookupError	0	0	1	0	0	0
ModuleNotFoundError	4	4	1	0	2	1
NameError	0	0	2	8	4	6
NotImplementedError	0	0	0	0	2	0
OSError	0	0	0	0	0	2
RuntimeError	0	0	0	0	0	0
SyntaxError	0	0	1	21	12	7
TypeError	7	24	4	4	4	13
UnboundLocalError	0	0	0	0	0	0
ValueError	15	26	33	10	5	11
EOFError	0	0	0	0	0	0
Python Library Errors						
DatasetNotFoundError	0	0	0	0	0	0
NotFoundError	0	0	0	1	0	0
OutOfMemory	0	0	1	0	3	0
ArgumentError	0	0	0	0	8	0
ScannerError	0	0	0	0	0	0
Other Python Errors						
ConstructorError	0	0	0	0	0	0
JSONDecodeError	0	0	1	0	0	0
HFValidationError	0	0	1	0	0	0
ParserError	0	0	1	0	0	0
Bash Errors						
cannot create directory	0	0	0	0	0	0
empty patch	4	0	1	15	17	22
empty or missing	0	2	0	5	10	7
unable to write file	0	0	0	0	0	0
Permission denied	0	0	0	2	0	1
syntax error	4	0	0	0	0	0
cannot access	0	1	0	0	0	0
patch failed	0	0	0	0	0	0
Execution Timeout	0	0	2	8	3	2

Table 9: Breakdown of error counts for OpenHands.

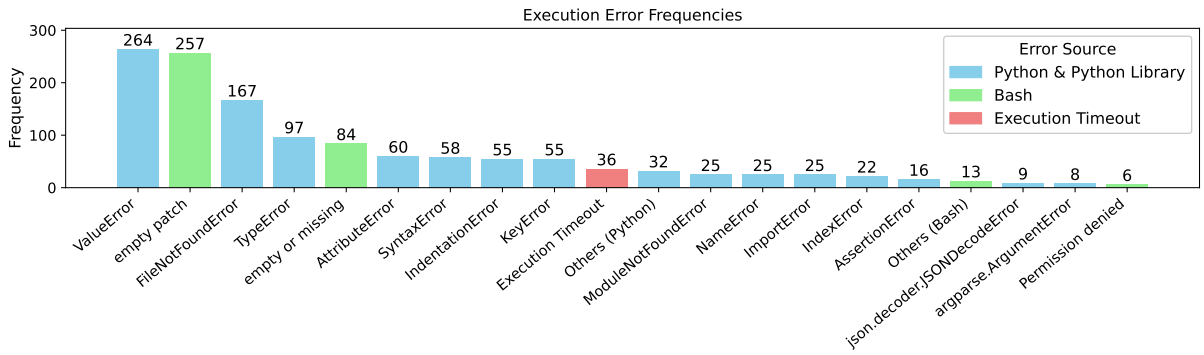

Figure 7: Distribution of execution errors across Python, Bash, and timeout categories. Errors with fewer than 5 occurrences are grouped as 'Others'.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	Claude 4 Sonnet	No Hints	CheckEval	0.00	1.00	1.00
			COGS	0.00	0.00	0.50
			Entity Tracking	0.60	0.80	1.00
			Explain then Translate	0.00	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	1.00	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.80	0.52
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.60	0.75
		Hints	CheckEval	0.00	1.00	1.00
			COGS	0.20	0.20	0.50
			Entity Tracking	0.60	0.80	1.00
			Explain then Translate	0.00	0.40	0.80
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.80	1.00	1.00
			Othello	0.80	0.80	1.00
			Re-reading	0.00	0.60	0.67
			Reasoning or Reciting	0.00	0.40	0.60
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.20	0.90
			WinoDict	0.00	0.20	0.75
		Detailed Hints	CheckEval	0.00	0.80	1.00
			COGS	0.00	0.00	1.00
			Entity Tracking	0.00	0.20	1.00
			Explain then Translate	0.80	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.40	1.00	1.00
			Othello	0.80	1.00	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.40	0.60
			Tree Of Thoughts	0.00	0.00	0.53
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.60	0.75

Table 10: Detailed performance on aider + Claude 4 Sonnet.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	Claude 3.7 Sonnet	No Hints	CheckEval	0.00	0.00	1.00
			COGS	0.00	0.00	0.50
			Entity Tracking	0.20	0.20	1.00
			Explain then Translate	0.00	0.60	1.00
			Instruction Tuning	0.00	0.40	0.00
			Mission Impossible	0.20	0.60	1.00
			Othello	0.60	0.60	1.00
			Re-reading	0.00	0.20	0.67
			Reasoning or Reciting	0.00	0.00	0.56
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.20	0.75
		Hints	CheckEval	0.00	0.00	1.00
			COGS	0.00	0.00	0.50
			Entity Tracking	0.20	0.40	1.00
			Explain then Translate	0.00	0.60	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.40	0.60	1.00
			Othello	0.60	0.60	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.00	0.75
		Detailed Hints	CheckEval	0.00	0.40	1.00
			COGS	0.00	0.00	0.60
			Entity Tracking	0.00	0.00	1.00
			Explain then Translate	0.60	0.60	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.60	1.00
			Othello	0.00	0.60	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.44
			Tree Of Thoughts	0.00	0.00	0.80
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.20	0.75

Table 11: Detailed performance on aider + Claude 3.7 Sonnet.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	OpenAI GPT-5	No Hints	CheckEval	0.00	0.60	0.80
			COGS	0.00	0.60	1.00
			Entity Tracking	0.00	0.60	1.00
			Explain then Translate	0.00	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.40	1.00
			Othello	0.00	0.80	1.00
			Re-reading	0.00	0.20	0.13
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.00	0.00	0.60
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.20	0.80	0.60
		Hints	CheckEval	0.00	0.60	1.00
			COGS	0.00	0.60	1.00
			Entity Tracking	0.00	0.60	1.00
			Explain then Translate	0.00	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.20	1.00
			Othello	0.00	1.00	1.00
			Re-reading	0.00	0.20	0.13
			Reasoning or Reciting	0.00	0.20	0.56
			Tree Of Thoughts	0.00	0.00	0.47
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.20	0.75
		Detailed Hints	CheckEval	0.00	0.20	1.00
			COGS	0.00	1.00	1.00
			Entity Tracking	0.00	0.60	0.80
			Explain then Translate	0.80	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.20	0.60	1.00
			Othello	0.00	0.00	1.00
			Re-reading	0.00	0.00	0.13
			Reasoning or Reciting	0.00	0.40	0.48
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.20	0.75

Table 12: Detailed performance on aider + OpenAI GPT-5.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	OpenAI o1	No Hints	CheckEval	0.00	0.40	0.80
			COGS	0.00	0.00	1.00
			Entity Tracking	0.00	1.00	1.00
			Explain then Translate	0.00	0.20	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.20	1.00
			Othello	0.00	0.00	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.36
			Tree Of Thoughts	0.00	0.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.60	0.75
		Hints	CheckEval	0.00	0.80	0.80
			COGS	0.00	0.00	1.00
			Entity Tracking	0.00	1.00	1.00
			Explain then Translate	0.00	0.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.60	1.00
			Othello	0.00	0.20	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.60
			Tree Of Thoughts	0.00	0.60	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.20	0.65
		Detailed Hints	CheckEval	0.00	0.00	0.80
			COGS	0.00	0.60	1.00
			Entity Tracking	0.00	1.00	1.00
			Explain then Translate	0.20	0.20	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.60	1.00
			Othello	0.00	1.00	1.00
			Re-reading	0.00	0.20	0.67
			Reasoning or Reciting	0.00	0.00	0.48
			Tree Of Thoughts	0.00	0.00	0.60
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.60	0.75

Table 13: Detailed performance on aider + OpenAI o1.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	OpenAI o4-mini	No Hints	CheckEval	0.00	0.00	0.60
			COGS	0.00	0.00	0.00
			Entity Tracking	0.00	0.60	0.80
			Explain then Translate	0.00	0.40	0.80
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.40	0.40	0.90
			Othello	0.00	0.40	1.00
			Re-reading	0.00	0.00	0.00
			Reasoning or Reciting	0.00	0.00	0.16
			Tree Of Thoughts	0.00	0.40	0.60
			VariErr-NLI	0.00	0.00	0.20
			WinoDict	0.00	0.20	0.30
		Hints	CheckEval	0.00	0.00	0.00
			COGS	0.00	0.00	0.10
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	0.00	0.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.40	0.80	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.00	0.00
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.00	0.00	0.13
			VariErr-NLI	0.00	0.00	0.40
			WinoDict	0.00	0.40	0.65
		Detailed Hints	CheckEval	0.00	0.00	0.40
			COGS	0.00	0.60	0.30
			Entity Tracking	0.80	1.00	1.00
			Explain then Translate	0.40	0.80	0.80
			Instruction Tuning	0.00	0.40	0.00
			Mission Impossible	0.00	0.40	0.80
			Othello	0.80	0.80	0.80
			Re-reading	0.00	0.00	0.00
			Reasoning or Reciting	0.00	0.00	0.44
			Tree Of Thoughts	0.00	0.00	0.33
			VariErr-NLI	0.00	0.00	0.80
			WinoDict	0.00	0.00	0.40

Table 14: Detailed performance on aider + OpenAI o4-mini.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
aider	DeepSeek-R1	No Hints	CheckEval	0.00	0.00	0.00
			COGS	0.00	0.00	0.00
			Entity Tracking	0.00	0.00	0.20
			Explain then Translate	0.00	0.00	0.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.40
			Othello	0.00	0.00	0.20
			Re-reading	0.00	0.00	0.13
			Reasoning or Reciting	0.00	0.00	0.00
			Tree Of Thoughts	0.00	0.00	0.00
			VariErr-NLI	0.00	0.00	0.20
			WinoDict	0.00	0.00	0.00
		Hints	CheckEval	0.00	0.00	0.20
			COGS	0.00	0.00	0.00
			Entity Tracking	0.00	0.00	0.00
			Explain then Translate	0.00	0.00	0.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.00
			Othello	0.00	0.00	0.20
			Re-reading	0.00	0.00	0.00
			Reasoning or Reciting	0.00	0.00	0.00
			Tree Of Thoughts	0.00	0.00	0.00
			VariErr-NLI	0.00	0.00	0.20
			WinoDict	0.00	0.00	0.00
		Detailed Hints	CheckEval	0.00	0.00	0.00
			COGS	0.00	0.00	0.20
			Entity Tracking	0.00	0.00	0.20
			Explain then Translate	0.00	0.00	0.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.00
			Othello	0.00	0.00	0.00
			Re-reading	0.00	0.00	0.00
			Reasoning or Reciting	0.00	0.00	0.08
			Tree Of Thoughts	0.00	0.00	0.00
			VariErr-NLI	0.00	0.00	0.40
			WinoDict	0.00	0.00	0.00

Table 15: Detailed performance on aider + DeepSeek-R1.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	Claude 4 Sonnet	No Hints	CheckEval	0.00	0.80	0.40
			COGS	1.00	1.00	0.60
			Entity Tracking	0.00	0.00	1.00
			Explain then Translate	0.60	1.00	1.00
			Instruction Tuning	0.40	0.40	0.00
			Mission Impossible	0.00	0.60	0.80
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.80	1.00
			Reasoning or Reciting	0.00	0.60	0.40
			Tree Of Thoughts	0.40	1.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.60	1.00	0.25
		Hints	CheckEval	0.00	0.20	0.80
			COGS	0.60	0.60	0.50
			Entity Tracking	0.80	0.80	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.60	0.80	0.00
			Mission Impossible	0.80	1.00	1.00
			Othello	0.80	0.80	1.00
			Re-reading	0.00	0.80	0.93
			Reasoning or Reciting	0.00	0.60	0.40
			Tree Of Thoughts	0.40	0.80	0.67
			VariErr-NLI	0.00	0.20	1.00
			WinoDict	0.20	0.60	0.65
		Detailed Hints	CheckEval	0.40	0.80	1.00
			COGS	0.80	0.80	1.00
			Entity Tracking	0.20	1.00	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.20	0.20	0.00
			Mission Impossible	0.60	1.00	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	1.00	0.87
			Reasoning or Reciting	0.00	1.00	0.40
			Tree Of Thoughts	0.80	1.00	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.20	0.60	0.75

Table 16: Detailed performance on OpenHands + Claude 4 Sonnet.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	Claude 3.7 Sonnet	No Hints	CheckEval	0.00	0.00	0.50
			COGS	0.80	1.00	0.50
			Entity Tracking	0.00	0.00	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.40	0.80	0.00
			Mission Impossible	0.00	0.00	0.50
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.40	0.73
			Reasoning or Reciting	0.00	0.00	0.36
			Tree Of Thoughts	0.00	0.40	0.67
			VariErr-NLI	0.00	0.40	1.00
			WinoDict	0.00	0.40	0.25
		Hints	CheckEval	0.40	0.80	0.80
			COGS	1.00	1.00	1.00
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	0.80	1.00	1.00
			Instruction Tuning	0.40	0.40	0.00
			Mission Impossible	0.00	0.60	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.20	0.60	0.67
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.00	0.70
		Detailed Hints	CheckEval	0.00	0.40	1.00
			COGS	1.00	1.00	1.00
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.00	0.60	0.00
			Mission Impossible	1.00	1.00	1.00
			Othello	0.00	1.00	1.00
			Re-reading	0.00	0.00	0.67
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.20	0.20	0.60
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.00	0.75

Table 17: Detailed performance on OpenHands + Claude 3.7 Sonnet.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	OpenAI GPT-5	No Hints	CheckEval	0.00	1.00	0.50
			COGS	0.80	0.80	0.50
			Entity Tracking	0.00	0.00	1.00
			Explain then Translate	0.60	0.80	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.20	0.60	0.80
			Othello	1.00	1.00	1.00
			Re-reading	0.00	0.60	0.93
			Reasoning or Reciting	0.00	0.80	0.40
			Tree Of Thoughts	0.20	1.00	0.67
			VariErr-NLI	0.00	0.40	1.00
			WinoDict	0.40	0.40	0.30
		Hints	CheckEval	0.00	0.20	1.00
			COGS	1.00	1.00	0.50
			Entity Tracking	0.40	0.60	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.20	1.00
			Othello	0.60	0.60	1.00
			Re-reading	0.00	0.80	0.93
			Reasoning or Reciting	0.00	1.00	0.40
			Tree Of Thoughts	0.80	1.00	0.67
			VariErr-NLI	0.00	0.40	1.00
			WinoDict	0.60	0.80	0.75
		Detailed Hints	CheckEval	0.40	0.60	0.90
			COGS	1.00	1.00	0.50
			Entity Tracking	0.40	0.40	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.20	0.40	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	1.00	1.00
			Reasoning or Reciting	0.00	1.00	0.40
			Tree Of Thoughts	0.80	0.80	0.67
			VariErr-NLI	0.00	0.20	1.00
			WinoDict	0.40	1.00	0.75

Table 18: Detailed performance on OpenHands + OpenAI GPT-5.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	OpenAI o1	No Hints	CheckEval	0.00	1.00	0.50
			COGS	0.00	0.00	0.40
			Entity Tracking	0.00	0.40	1.00
			Explain then Translate	0.00	1.00	0.80
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.50
			Othello	0.00	0.00	0.80
			Re-reading	0.00	0.40	0.67
			Reasoning or Reciting	0.00	0.00	0.24
			Tree Of Thoughts	0.00	0.40	0.53
			VariErr-NLI	0.00	0.00	0.80
			WinoDict	0.00	0.80	0.25
		Hints	CheckEval	0.00	1.00	0.70
			COGS	0.00	0.00	0.50
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	0.00	0.40	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.60	0.50
			Othello	0.00	0.00	1.00
			Re-reading	0.00	0.40	0.73
			Reasoning or Reciting	0.00	0.00	0.24
			Tree Of Thoughts	0.00	0.60	0.60
			VariErr-NLI	0.00	0.00	0.70
			WinoDict	0.00	0.00	0.65
		Detailed Hints	CheckEval	0.00	0.00	0.80
			COGS	0.00	1.00	0.90
			Entity Tracking	0.00	0.60	0.80
			Explain then Translate	0.40	0.60	0.60
			Instruction Tuning	0.00	0.40	0.00
			Mission Impossible	0.00	0.00	0.80
			Othello	0.00	0.40	0.90
			Re-reading	0.00	0.40	0.93
			Reasoning or Reciting	0.00	0.00	0.40
			Tree Of Thoughts	0.00	1.00	0.60
			VariErr-NLI	0.00	0.00	0.70
			WinoDict	0.00	0.20	0.45

Table 19: Detailed performance on OpenHands + OpenAI o1.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	OpenAI o4-mini	No Hints	CheckEval	0.00	0.40	0.50
			COGS	0.20	0.20	0.50
			Entity Tracking	0.40	1.00	1.00
			Explain then Translate	0.00	0.60	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.10
			Othello	0.40	0.40	0.90
			Re-reading	0.00	0.60	1.00
			Reasoning or Reciting	0.00	0.40	0.40
			Tree of Thoughts	0.00	0.60	0.00
			VariErr-NLI	0.00	0.00	0.80
			WinoDict	0.00	0.20	0.25
		Hints	CheckEval	0.00	0.00	0.50
			COGS	0.80	0.80	0.50
			Entity Tracking	0.80	1.00	1.00
			Explain then Translate	0.00	0.20	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.80	0.70
			Othello	0.60	0.60	0.90
			Re-reading	0.40	1.00	0.93
			Reasoning or Reciting	0.00	0.00	0.32
			Tree of Thoughts	0.00	0.00	0.00
			VariErr-NLI	0.00	0.00	0.90
			WinoDict	0.00	0.00	0.70
		Detailed Hints	CheckEval	0.00	0.40	0.60
			COGS	0.60	0.60	0.50
			Entity Tracking	0.00	0.80	0.80
			Explain then Translate	0.00	0.80	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.60	1.00
			Othello	1.00	1.00	1.00
			Re-reading	0.00	1.00	1.00
			Reasoning or Reciting	0.00	0.60	0.40
			Tree of Thoughts	0.00	0.00	0.00
			VariErr-NLI	0.00	0.00	0.60
			WinoDict	0.00	0.00	0.60

Table 20: Detailed performance on OpenHands + OpenAI o4-mini.

Agent	Model	Hint Level	Task	Final Success	Execution Success	File Recall
OpenHands	DeepSeek-R1	No Hints	CheckEval	0.00	0.00	0.40
			COGS	0.00	0.40	0.40
			Entity Tracking	0.00	0.00	0.60
			Explain then Translate	0.00	0.40	1.00
			Instruction Tuning	0.00	0.40	0.00
			Mission Impossible	0.00	0.00	0.30
			Othello	0.00	0.40	1.00
			Re-reading	0.00	0.00	0.40
			Reasoning or Reciting	0.00	0.00	0.28
			Tree Of Thoughts	0.00	0.40	0.67
			VariErr-NLI	0.00	0.00	0.80
			WinoDict	0.00	0.00	0.15
		Hints	CheckEval	0.00	0.20	0.50
			COGS	0.00	0.00	0.50
			Entity Tracking	0.00	0.00	0.80
			Explain then Translate	0.00	0.60	0.80
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	0.60
			Othello	0.00	0.40	0.80
			Re-reading	0.00	0.00	0.60
			Reasoning or Reciting	0.00	0.00	0.44
			Tree Of Thoughts	0.00	0.00	0.47
			VariErr-NLI	0.00	0.40	1.00
			WinoDict	0.00	0.00	0.30
		Detailed Hints	CheckEval	0.00	0.00	0.60
			COGS	0.00	0.20	0.50
			Entity Tracking	0.00	0.60	0.80
			Explain then Translate	0.80	1.00	1.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.00	0.00	1.00
			Othello	0.40	0.40	0.80
			Re-reading	0.00	0.00	0.93
			Reasoning or Reciting	0.00	0.00	0.32
			Tree Of Thoughts	0.00	0.00	0.53
			VariErr-NLI	0.00	0.00	1.00
			WinoDict	0.00	0.40	0.15

Table 21: Detailed performance on OpenHands + DeepSeek-R1.

Task	Repository	License
CheckEval	yukyunglee/CheckEval	MIT
COGS	najoungkim/COGS	MIT
Entity Tracking	najoungkim/code-models-entity-tracking	Apache 2.0
Explain then Translate	PootieT/explain-then-translate	MIT
Instruction Tuning	john-hewitt/implicit-ins	Apache 2.0
Mission Impossible	jkallini/mission-impossible-language-models	???
Othello	likenneth/othello_world	MIT
Reasoning or Reciting	ZhaofengWu/counterfactual-evaluation	MIT
Re-reading	EleutherAI/lm-evaluation-harness	MIT
Tree of Thoughts	princeton-nlp/tree-of-thought-llm	MIT
VariErr-NLI	mainlp/VariErr-NLI	MIT
WinoDict	google-research/language/tree/master/language/wino_dict	Apache 2.0

Table 22: Licenses for each Github repository.