

000 **REXBENCH: CAN CODING AGENTS AUTONOMOUSLY** 001 **IMPLEMENT AI RESEARCH EXTENSIONS?**

002 **Anonymous authors**

003 Paper under double-blind review

004 ABSTRACT

005 Agents based on Large Language Models (LLMs) have shown promise for per-
006 forming sophisticated software engineering tasks autonomously. In addition, there
007 has been progress towards developing agents that can perform parts of the research
008 pipeline in machine learning and the natural sciences. We argue that research
009 *extension* and its implementation is a critical capability for such systems, and intro-
010 duce **REXBENCH** to support the evaluation of this capability. **REXBENCH** is a
011 benchmark consisting of realistic extensions of 12 research papers that aim to investi-
012 gate *novel* research hypotheses. Each task is set up as an extension to an existing
013 research paper and codebase, accompanied by domain expert-written instructions.
014 **REXBENCH** is robust to data contamination, and supports an automatic evaluation
015 infrastructure that executes agent outputs to determine whether the success criteria
016 are met. We use this benchmark to evaluate 13 LLM agents implemented using
017 three different frameworks: aider, Claude Code, and OpenHands. We find that
018 all agents fail to autonomously implement the majority of the extensions, with
019 the best agent at around 31% success rate. Although the success rate improves
020 with additional human-written hints, the best performance under this setting re-
021 mains below 48%. This indicates that current agents are still short of being able
022 to handle realistic research extension tasks without substantial human guidance.
023 Based on analyses of prominent failure modes, we put forward actionable short-
024 and long-horizon recommendations for future research coding agent development.

025 1 INTRODUCTION

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

046 We tested thirteen agents based on an array of Large Language Model (LLM) backbones (Claude
047 4/3.7 Sonnet (Anthropic, 2025; 2024), GPT-5 (OpenAI, 2025), o1 (Jaech et al., 2024), o4-mini,
048 and DeepSeek R1 (Guo et al., 2025)), using three different agent frameworks (aider, Claude Code,

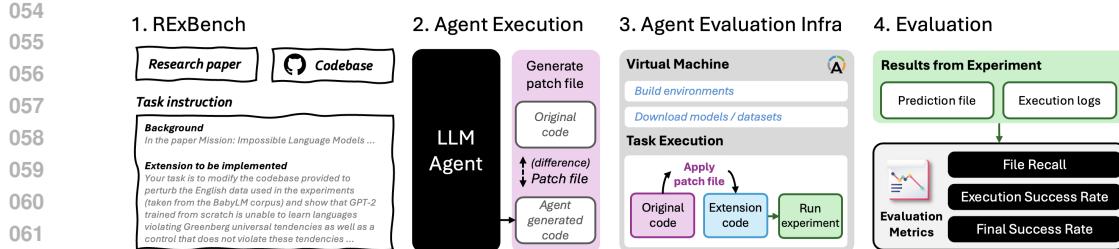


Figure 1: End-to-end workflow of **REXBENCH**: (1) An LLM agent receives inputs consisting of the research paper(s), the original codebase, and an extension instruction; (2) the system implements the extension and a patch file is obtained; (3) the patch is applied to the original code and executed via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

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, often showing qualitative signs of success even when they did not achieve final success. Nevertheless, even the best-performing agents succeeded less than one third of the time on average (31% success rate for OpenHands + {Claude 4 Sonnet, GPT-5}), 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 motivated discussions about their applicability to scientific research. This includes using LLMs and LLM-based agents for research automation (Li et al., 2025; 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) 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). For ML research, current attempts span automation across all stages of the research process: from ideation (Si et al., 2024) to experiment design (Abramovich & Chechik, 2025) and execution (Siegel et al., 2024; Xiang 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).

Another line of work benchmarks coding and software engineering skills. Specific skills targeted include resolving GitHub issues (SWE-Bench, Jimenez et al., 2024, SWE-rebench, Badertdinov et al., 2025), debugging LeetCode problems (DebugBench, Tian et al., 2024b), resolving configuration/dependency issues in research environment setups (SUPER, 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 such as regression and model training problems as well as Kaggle challenges; DataSciBench evaluates data analysis and visualization skills with novel evaluation pipelines (Zhang et al., 2025); and ML-Dev-Bench (Padigela et al., 2025) focuses on the full ML development workflow.

The most directly relevant efforts to ours are benchmarks that evaluate 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. Paper2Code (Seo et al., 2025) introduces a multi-agent LLM framework to translate ML papers into codebases through a stage-wise design, and PaperBench (Starace et al., 2025) evaluates research agents using a compilation of

108 coding tasks targeting replication of 20 ICML papers. **REXBENCH** has a similar goal as PaperBench
 109 and, to some extent, Curie, in benchmarking of ML and AI research code generation. However, a key
 110 distinction is that instead of evaluating replications (PaperBench) or very general questions that can
 111 often also be answered without running experiments (Curie), we focus on *novel research extensions*.
 112 Thus, **REXBENCH** is able to evaluate agent performance on previously unseen/unimplemented
 113 research hypotheses which greatly alleviates data contamination concerns.

114

115 3 BENCHMARK DESIGN

116 3.1 RESEARCH EXTENSION TASK

117 **Task** We define our research extension task as a code implementation problem, where the input
 118 consists of an existing research paper, an accompanying codebase, and an instruction that verbally
 119 describes an extension proposal and how this should be tested. An example of a simple extension
 120 is: “What would happen if the same experiment in paper X used an open-source model like Llama 3
 121 70B instead of GPT-4o?” (see Appendix C for an actual example). Given this input, a system must
 122 produce as output edits to the input codebase that implements the extension proposal.

123

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

147

148 3.2 BENCHMARK COMPOSITION

149 Our benchmark consists of research extensions building upon papers and codebases primarily in
 150 the NLP and broader AI domains, taking into consideration the availability of expertise within
 151 the team as well as the availability/replicability of the code released. The full list of papers is in
 152 Table 1.² The specific extension proposals were selected to span various dimensions of change
 153 including changes to the model, dataset, algorithm, and evaluation. In addition to this consideration,
 154 we imposed the following constraints on the extension proposals for scientific rigor and feasibility
 155 of the experiments: (1) Important empirical trends from the original paper relevant to the extension
 156 proposal must replicate; (2) the gold implementation of the extension proposal must replicate (e.g., if
 157 the gold implementation requires making calls to a closed API-based model, this may not replicate

158

¹We 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.

159

160

161

²Two 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.

162 in the future due to model deprecation); and (3) the estimated runtime of each gold implementation
 163 should be shorter than 12 hours on a single A100 GPU. The final dataset includes the extension
 164 instruction, target research papers in both .pdf and .md format (converted using PyMuPDF 4LLM to
 165 accommodate agents that lack the ability to read .pdf files), and the original codebase.
 166

167 3.3 BENCHMARK CONSTRUCTION PROCESS

169 For each extension proposal, a domain expert (PhD student-level or above) first verified that the
 170 original codebase replicates the results of the associated paper on our virtual machines (details to
 171 follow). Then, they implemented the “gold” edits for the target extension and recorded the numerical
 172 outcomes, ensuring that the runtime does not exceed 12 hours. This implementation process and
 173 the outcomes were validated by at least one other author. Finally, the domain expert wrote the
 174 instruction that consists of a brief description of the original paper, the extension proposal, and how
 175 this proposal should be tested (see Appendix C for a full instruction for one of our tasks, WinoDict).
 176 The description of the “how” was deliberately high-level to meet the desideratum of evaluating a
 177 sufficiently autonomous capacity. Nevertheless, since the instructions should not be confusing or
 178 ambiguous, they were polished through multiple rounds of revisions by multiple authors to improve
 179 clarity. Importantly, if we foresaw degrees of implementation freedom that may introduce random
 180 variation, we controlled for this by specifying constraints (e.g., use an implementation of Pearson
 181 correlation function from the `scipy` package as opposed to implementing this from scratch). During
 182 this revision process we furthermore ensured that each extension was self-contained. No part of
 183 the gold edits required information external to the set of inputs provided to the system. As a part
 184 of the revisions for self-containment, we provided information such as specific model identifiers
 185 and explanations of necessary hyperparameters not in any README or the paper as a part of the
 186 instruction, and added version information for all of the packages (via an `environment.yml` file).
 187

188 3.4 EVALUATION METRICS

189 Our main metric is final success rate, which measures whether the outcome of executing the model-
 190 implemented code falls within the target range (more details below). We define two additional metrics
 191 for finer-grained analyses: execution success rate and file recall. We describe each metric below.
 192

193 **Final Success Rate** Final success rate evaluates whether the model correctly implements the
 194 specified research extension. This evaluation either checks whether the final results exactly match
 195 the results of the reference implementation (if the run is fully deterministic) or it checks whether the
 196 results fall within a *gold range* that we obtained by running the gold implementation multiple times
 197 with different random seeds to account for output variability. In the latter case, a model solution is
 198 considered successful if its final execution outcome falls within this bound.
 199

200 **Execution Success Rate** Execution success rate checks whether the generated code runs without
 201 errors in our evaluation environment. This metric evaluates the general well-formedness of the code
 202 and contextual understanding sufficient to avoid runtime errors.
 203

204 **File Recall** File recall quantifies whether files edited in the gold solution were also edited by
 205 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
 206 dependency on the gold solution. Technically, a solution could achieve zero file recall with perfect
 207 final success. E.g., if a model solution was exactly equivalent to gold but created new files with
 208 identical content instead of editing, and changed references appropriately in the repository, this would
 209 be the case. Still, we take human expert edits to reflect a reasonably efficient set of modifications.
 210

211 3.5 EVALUATION INFRASTRUCTURE

212 **Submission format** Our metrics defined above require execution of agent generated code. We
 213 propose to conduct this execution on a virtual machine to control for hardware specification and
 214 package dependencies. We will host this infrastructure using our own resources, and conduct
 215 evaluation asynchronously at a regular interval to update the leaderboard with the submissions we
 receive, similarly to Jimenez et al. (2024). The submissions will be received in the form of git patch



Figure 2: Agent performance on **REXBENCH**. The color coding indicates the agent framework and the y axis indicates the the backbone LLM. Results include three 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).

files (as opposed to full edited repositories) to streamline the submission process. Additionally, we will 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 launch an Ubuntu virtual machine instance with a 20GB root disk, where we run a task-specific Apptainer container (Singularity, 2021) that has the original codebase and evaluation scripts pre-loaded and the environment set up. Each instance is also equipped with task-specific hardware: either a single NVIDIA A100 40GB GPU, a single K80 12GB GPU, or just a CPU (see Appendix B, Table 2). 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. Inside the container, we apply the patch file and execute a single bash script `run_apptainer.sh` that contains all necessary commands (this requirement is also provided in the task instructions). 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 2 for all estimated runtimes). Once task execution is complete or the attempt crashes, any result files and task execution logs are copied to an external storage volume. We then delete the virtual machine instance and evaluate the results. 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 thirteen LLM agents, combining three 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 three times with the same agent model to account for agent random variation.

4.1.1 BASELINE AGENT DESIGN

We used three different agent frameworks: two open-source (aider, aider AI (2023) and OpenHands, Wang et al. (2025)) that we adapted for the task, as well as the proprietary Claude Code. aider and OpenHands both support several 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 to the best of our effort. 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

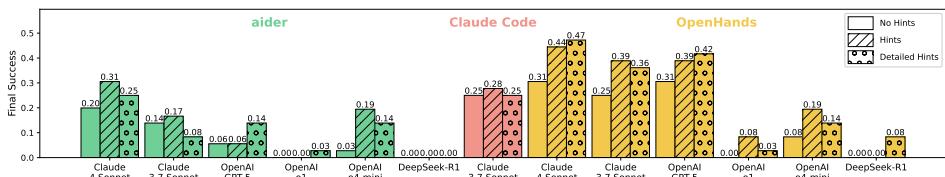


Figure 3: Final success rates for each agent-LLM combination and hint level.

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 D.

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 layer of sanity 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 mainly 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 directly 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.

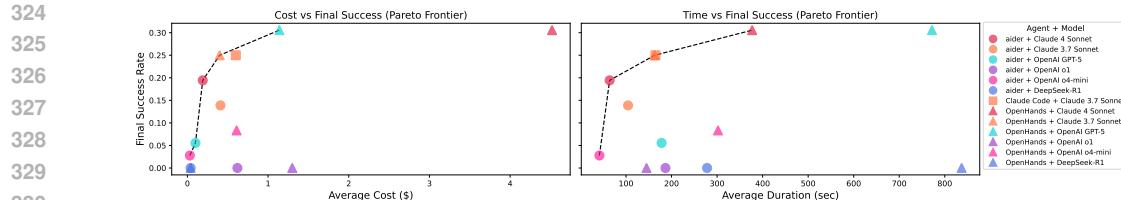
4.3 RESULTS

Main experiment Figure 2 shows our main results. Most agents struggle with the task, with the best performing agents (OpenHands + {Claude 4 Sonnet, GPT-5}) achieving 31% average final success rates. Claude 4 Sonnet and GPT-5 were the best backbone LLMs—when different LLMs were available, they yielded the strongest performance. All agents achieved nonzero execution success rates except for DeepSeek-R1, which failed completely. Claude 4 Sonnet and GPT-5 again performed best, with success rates of 67% and 72% respectively when combined with 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 4 in Appendix F) show 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 the best agents, OpenHands + {Claude 4 Sonnet, GPT-5}, achieving 47% and 42% final success rates, respectively.

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 and OpenHands + Claude 3.7 Sonnet lie on the Pareto frontier for both cost and time. We provide the full time and cost estimates for agent runs in Appendix F, Table 6. In terms of token usage statistics, aider consistently used 2 turns due to its non-iterative design. Claude Code used 25–35 turns and OpenHands used 17–93 turns, making use of active multi-turn structures. Due to its closed-source nature, we could not obtain token counts

331 Figure 4: Cost effectiveness and time efficiency of coding agents on **REXBENCH**.
332
333

334 for Claude Code.³ OpenHands used the most tokens, especially with Claude 4 Sonnet and GPT-5,
335 reaching up to 1.86M prompt tokens (almost 560 times more than aider). As the hint levels increased,
336 both turns and token usage in OpenHands tended to grow, while the turns in Claude Code decreased.
337 See Table 5 in Appendix F for token usage statistics by model and by hint levels.

339 5 ANALYSIS AND DISCUSSION

340 5.1 PATTERNS OF ERROR

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

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

357 **Implicit errors** Analysis of implicit errors (execution success but mismatch with gold outcome)
358 involved greater manual effort because it required a holistic review of agent edits. Therefore, we
359 focused our analysis on the top 2 agents (OpenHands + {Claude 4, GPT-5}). Overall, the agents'
360 implicit errors were categorizable into errors in implementation logic and errors in value (e.g.,
361 within-bounds index errors, incorrect hyperparameters or paths)—the ratio of logic vs. value errors
362 was about 2:1 for Claude 4 and about 1:1 for GPT-5. We also estimated the debugging difficulty
363 from the manually identified sources of error, using the scale of easy (requires small local fix),
364 medium (requires logical but local revisions), and hard (requires holistic revisions). For both models,
365 the majority of the errors were easy to debug. OpenHands + GPT-5 had more implicit errors,
366 especially ones falling into the easy and medium categories (21 easy, 12 medium, 3 hard) compared
367 to OpenHands + Claude 4 (16 easy, 4 medium, 4 hard), revealing a qualitative difference in the agents'
368 solutions although the quantitative success rates were similar. Many of the medium and hard implicit
369 errors arose from agents “over-editing” the code beyond the given instructions (e.g., adding extra
370 (incorrect) exception handling or changing irrelevant flags/prompts). These unrequested edits often
371 caused silent failures or subtle deviations from the gold implementation leading to markedly different
372 results, making debugging harder. Task-specific examples are discussed in Appendix G.

373
374
375
376
377 ³As of now, token counts for Claude Code have become available, but this feature was not available when
378 our experiments were conducted.

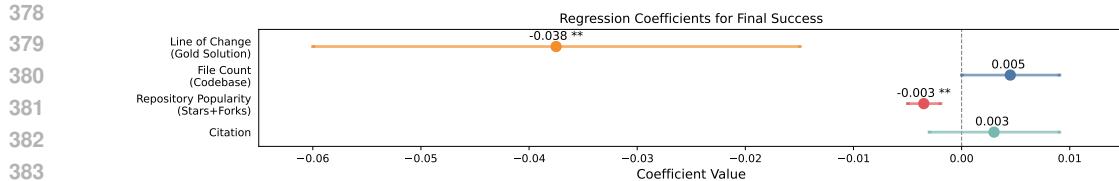


Figure 5: Regression coefficients with 95% confidence intervals for predictors of final success. (Regression model: $\text{final_success} \sim \text{line_change} + \text{file_count} + \text{repository_popularity} + \text{citation} + (1 | \text{model})$). (*: $p < .05$, **: $p < .01$)

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 heavy sanity checks (perhaps supported by system design) if an agent were to be deployed 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 agents (OpenHands + {Claude 4, GPT-5}) benefited from both levels of hints, many others showed no improvement. The best agents benefitted 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 and Claude Code 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, these agents employed a qualitatively different strategy with the second level of hints. However, it was not the case that this particular hint was misleading, since the 2 best performing agents were 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. We noticed this pattern for two tasks (Othello and Tree-of-Thoughts), but not in general.

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.

432 **6 CONCLUSION**

434 We presented **REXBENCH**, a benchmark evaluating the autonomous capacity of AI systems to
 435 implement hypothesis-driven research extensions in the domain of AI research. **REXBENCH** consists
 436 of realistic but well-scoped extension tasks motivated by existing research. To perform well, a system
 437 must be able to understand the expert-written extension instructions situated in specific research
 438 context, understand the structure and logic of the original codebase, and autonomously plan and
 439 implement the requested extension. Our tasks are by design robust to data contamination due to the
 440 extensions requiring novel implementations whose solutions are not available publicly. Experiments
 441 with various agent frameworks combined with competent backbone LLMs show that most systems
 442 struggle on our benchmark, with the best performing models (OpenHands + {Claude 4 Sonnet,
 443 GPT-5}) achieving 31% extension success rate. Notably, agents with o1 or DeepSeek-R1 as backbone
 444 showed (close to) zero success rate. Nevertheless, closer analysis of the best models revealed promise:
 445 the strongest backbone LLMs (Claude 4 Sonnet, GPT-5) achieved higher execution success rates
 446 than weaker models, with implementations often syntactically valid and logically on the right track.
 447 This observation, taken together with the large headroom, highlights the utility of **REXBENCH** for
 448 guiding future developments of research agents. Finally, based on the analyses of the agents tested in
 449 this work, we put forward several actionable recommendations for the future.

450 **Short-horizon recommendations:**

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

464 **Longer-horizon recommendations:**

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

481 **The future of REXBENCH** Finally, as discussed in the introduction, we view the release of
 482 **REXBENCH** and this paper as a motivating start to a larger community-driven effort. While our
 483 tasks were primarily in the AI domain with a focus on topics aligning with the expertise of our team,
 484 we believe the format of the extension task and evaluation framework shown in Figure 1 are broadly
 485 applicable beyond our set of tasks. We hope the current set draws community interest in research
 486 extensions as an interesting problem for agents, and hope to collaborate with researchers and/or
 487 solicit community contributions for a comprehensive coverage of task domains and implementation
 488 complexities.

486 ETHICS STATEMENT
487488 In this work, we showed that current LLM-based agents cannot reliably produce code for AI research
489 without additional human supervision. We based this argument on the low final success rate of all
490 evaluated agents, as well as the danger of the increasing trend of implicit errors as model capacity
491 improves. Given the rapid progress of AI research and model development, it is a likely possibility
492 that new agents would perform significantly better on this benchmark in the near future. The biggest
493 risk we therefore foresee is that good performance on this benchmark is seen as a sufficient condition
494 for reliable agents rather than a necessary one. While we consider the benchmark to be well-suited
495 for measuring progress in the development of future agents, good performance should NOT be seen
496 as sufficient evidence for an agent being able to autonomously produce reliable research code.497 We also would like to highlight again that the baseline agents we evaluated for this work did not reach
498 the level of competence that would translate into autonomous research extension capacities in the real
499 world. Given the difficulty of debugging, deployment of such systems without rigorous verification
500 measures faces the danger of leading researchers to draw conclusions from faulty implementations
501 and of the erosion of trust in published results.502 Furthermore, as discussed in Section 3.1, a benchmark being realistic inherently conflicts with the
503 ease of automatic evaluation. In particular, a task like research extension can be extremely open
504 ended in reality, even when constrained with a specific proposal and hypothesis. We opted for a
505 middle ground where we do not enforce strong limits on *how* a system may implement the target
506 extension and condition final success only on alignment of numerical outcomes. This necessitated
507 a stronger control for sources of variation, which led us to write instructions as self-contained and
508 unambiguous as possible. This setting is idealized as the instructions are much more informative and
509 clearer than an actual task a human researcher faces, even in scenarios where the extension idea is
510 provided to them (e.g., an advisor suggesting to a PhD student “How about we try X?”), missing out
511 on the real difficulties lying in the initial trial-and-error concretization step.512 Finally, executing machine-written code always bears safety risks and providing AI agents with too
513 much freedom for exploration may enable them to cause harm. To mitigate this risk, we narrowly
514 scoped the implementation tasks in our experiments fully based on human-generated hypotheses and
515 instructions. Furthermore, any machine-written code was executed in a containerized environment
516 without internet access. We recommend similar setups for the execution of any code that is output by
517 AI agents.518
519 REPRODUCIBILITY STATEMENT
520521 Our dataset and the code for our baseline agents are submitted as supplementary material. We have
522 furthermore taken the following steps to ensure reproducibility of our experimental results. First, we
523 fix random seed values across multiple libraries, including Python’s built-in random module, NumPy,
524 PyTorch (CPU and CUDA), and CUDA cuDNN, in order to control for nondeterminism in obtaining
525 the execution outcomes from the gold solutions. Second, we execute both the gold solutions and
526 agent solutions within virtual machines using Apptainer containers. These containers are configured
527 with identical hardware resources and software/library versions, ensuring that all experiments run
528 under consistent conditions and that reported performance is not affected by hardware or software
529 variability. In addition, while we refrain from releasing gold solutions publicly due to contamination
530 concerns, we welcome interested researchers to request access to specific ground-truth solutions for
531 further validation and standardization purposes.532
533 REFERENCES
534535 Talor Abramovich and Gal Chechik. Ablationbench: Evaluating automated planning of ablations in
536 empirical ai research. *arXiv preprint arXiv:2507.08038*, 2025.537
538 aider AI. aider: Ai pair programming in your terminal. <https://github.com/Aider-AI/aider>, 2023. Accessed: 2025-05-12.

540 Anthropic. Claude 3.7 sonnet system card. <https://assets.anthropic.com/m/785e231869ea8b3b/original/clause-3-7-sonnet-system-card.pdf>, 2024.
 541 Accessed: 2025-05-14.

542

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

544

545

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

550

551 Ben Beglin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish
 552 Sabharwal, and Tushar Khot. SUPER: Evaluating agents on setting up and executing tasks from
 553 research repositories. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Pro-
 ceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp.
 554 12622–12645, Miami, Florida, USA, November 2024. Association for Computational Linguis-
 555 tics. doi: 10.18653/v1/2024.emnlp-main.702. URL <https://aclanthology.org/2024.emnlp-main.702/>.

556

557 Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research
 558 with large language models. *Nature*, 624(7992):570–578, 2023. URL <https://www.nature.com/articles/s41586-023-06792-0>.

559

560

561 Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi
 562 Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language
 563 agents for data-driven scientific discovery, october 2024. *arXiv:2410.05080*, 2024. URL <https://arxiv.org/abs/2410.05080>.

564

565 Jonathan H. Choi. How to use large language models for empirical legal research. *Journal of
 566 Institutional and Theoretical Economics (JITE)*, 180(2):214–233, 2024. URL <https://ideas.repec.org/a/mhr/jinste/urndo10.1628-jite-2024-0006.html>.

567

568 Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. The devil is in the detail: Simple tricks
 569 improve systematic generalization of transformers. In Marie-Francine Moens, Xuanjing Huang,
 570 Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical
 571 Methods in Natural Language Processing*, pp. 619–634, Online and Punta Cana, Dominican
 572 Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.49. URL <https://aclanthology.org/2021.emnlp-main.49/>.

573

574

575 Jiangshu Du, Yibo Wang, Wenting Zhao, Zhongfen Deng, Shuaiqi Liu, Renze Lou, Henry Peng
 576 Zou, Pranav Narayanan Venkit, Nan Zhang, Mukund Srinath, Haoran Ranran Zhang, Vipul Gupta,
 577 Yinghui Li, Tao Li, Fei Wang, Qin Liu, Tianlin Liu, Pengzhi Gao, Congying Xia, Chen Xing,
 578 Cheng Jiayang, Zhaowei Wang, Ying Su, Raj Sanjay Shah, Ruohao Guo, Jing Gu, Haoran Li,
 579 Kangda Wei, Zihao Wang, Lu Cheng, Surangika Ranathunga, Meng Fang, Jie Fu, Fei Liu, Ruihong
 580 Huang, Eduardo Blanco, Yixin Cao, Rui Zhang, Philip S. Yu, and Wenpeng Yin. LLMs assist NLP
 581 researchers: Critique paper (meta-)reviewing. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung
 582 Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language
 583 Processing*, pp. 5081–5099, Miami, Florida, USA, November 2024. Association for Computational
 584 Linguistics. doi: 10.18653/v1/2024.emnlp-main.292. URL <https://aclanthology.org/2024.emnlp-main.292/>.

585

586 Julian Martin Eisenschlos, Jeremy R. Cole, Fangyu Liu, and William W. Cohen. WinoDict:
 587 Probing language models for in-context word acquisition. In Andreas Vlachos and Isabelle
 588 Augenstein (eds.), *Proceedings of the 17th Conference of the European Chapter of the As-
 589 sociation for Computational Linguistics*, pp. 94–102, Dubrovnik, Croatia, May 2023. Asso-
 590 ciation for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.7. URL <https://aclanthology.org/2023.eacl-main.7/>.

591

592 Kanishk Gandhi, Michael Y Li, Lyle Goodyear, Louise Li, Aditi Bhaskar, Mohammed Zaman, and
 593 Noah D Goodman. Boxinggym: Benchmarking progress in automated experimental design and
 model discovery. *arXiv:2501.01540*, 2025. URL <https://arxiv.org/abs/2501.01540>.

594 Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom
 595 Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an ai co-scientist.
 596 *arXiv:2502.18864*, 2025. URL <https://arxiv.org/abs/2502.18864>.

597

598 Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran
 599 Pan, Teng Wu, Jiaqian Yu, Yikun Zhang, Tianmai M. Zhang, Lanyi Zhu, Mike A Merrill, Jeffrey
 600 Heer, and Tim Althoff. BLADE: Benchmarking language model agents for data-driven science.
 601 In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for*
 602 *Computational Linguistics: EMNLP 2024*, pp. 13936–13971, Miami, Florida, USA, November
 603 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.815.
 604 URL <https://aclanthology.org/2024.findings-emnlp.815/>.

605 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
 606 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-R1: Incentivizing reasoning capability in
 607 LLMs via reinforcement learning. *arXiv:2501.12948*, 2025. URL <https://arxiv.org/abs/2501.12948>.

608

609 John Hewitt, Nelson F. Liu, Percy Liang, and Christopher D. Manning. Instruction following
 610 without instruction tuning. *arXiv:2409.14254*, 2024. URL <https://arxiv.org/abs/2409.14254>.

611

612

613 Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: evaluating language agents
 614 on machine learning experimentation. In *Proceedings of the 41st International Conference on*
 615 *Machine Learning*, pp. 20271–20309, 2024. URL <https://proceedings.mlr.press/v235/huang24y.html>.

616

617 Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Hel-
 618 yar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv:2412.16720*,
 619 2024. URL <https://arxiv.org/abs/2412.16720>.

620

621 Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi Mishra,
 622 Bodhisattwa Prasad Majumder, Daniel S. Weld, and Peter Clark. Codescientist: End-to-end
 623 semi-automated scientific discovery with code-based experimentation. *arXiv:2503.22708*, 2025.
 624 URL <https://arxiv.org/abs/2503.22708>.

625

626 Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R.
 627 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth*
 628 *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

629

630 Liqiang Jing, Zehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming
 631 Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents to becoming data
 632 science experts? *arXiv:2409.07703*, 2024. URL <https://arxiv.org/abs/2409.07703>.

633

634 Julie Kallini, Isabel Papadimitriou, Richard Futrell, Kyle Mahowald, and Christopher Potts. Mis-
 635 sion: Impossible language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar
 636 (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Lin-
 637 guistics (Volume 1: Long Papers)*, pp. 14691–14714, Bangkok, Thailand, August 2024. As-
 638 sociation for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.787. URL <https://aclanthology.org/2024.acl-long.787/>.

639

640 Naojoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic
 641 interpretation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of*
 642 *the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9087–
 643 9105, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.
 644 emnlp-main.731. URL <https://aclanthology.org/2020.emnlp-main.731/>.

645

646 Naojoung Kim, Sebastian Schuster, and Shubham Toshniwal. Code pretraining improves entity
 647 tracking abilities of language models. *arXiv:2405.21068*, 2024. URL <https://arxiv.org/abs/2405.21068>.

648 Hiroaki Kitano. Nobel turing challenge: creating the engine for scientific discovery. *NPJ systems*
 649 *biology and applications*, 7(1):29, 2021. URL <https://www.nature.com/articles/s41540-021-00189-3>.
 650

651 Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth
 652 Srinivas, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. Curie: Toward rigorous and
 653 automated scientific experimentation with ai agents. *arXiv:2502.16069*, 2025. URL <https://arxiv.org/abs/2502.16069>.
 654

655 Jon M. Laurent, Joseph D. Janizek, Michael Ruzo, Michaela M. Hinks, Michael J. Hammerling,
 656 Siddharth Narayanan, Manvitha Ponnappati, Andrew D. White, and Samuel G. Rodrigues. Lab-
 657 bench: Measuring capabilities of language models for biology research. *arXiv:2407.10362*, 2024.
 658 URL <https://arxiv.org/abs/2407.10362>.
 659

660 Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Pilsung Kang, and Najoung Kim.
 661 Checkeval: A reliable LLM-as-a-judge framework for evaluating text generation using checklists.
 662 *arXiv:2403.18771*, 2025. URL <https://arxiv.org/abs/2403.18771>.
 663

664 Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wat-
 665 tenberg. Emergent world representations: Exploring a sequence model trained on a synthetic
 666 task. In *The Eleventh International Conference on Learning Representations*, 2023. URL
 667 https://openreview.net/forum?id=DeG07_TcZvT.
 668

669 Sihang Li, Jin Huang, Jiaxi Zhuang, Yaorui Shi, Xiaochen Cai, Mingjun Xu, Xiang Wang, Linfeng
 670 Zhang, Guolin Ke, and Hengxing Cai. ScilitLLM: How to adapt LLMs for scientific literature
 671 understanding. In *The Thirteenth International Conference on Learning Representations*, 2025.
 672 URL <https://openreview.net/forum?id=8dzKkeWUUb>.
 673

674 Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai
 675 scientist: Towards fully automated open-ended scientific discovery. *arXiv:2408.06292*, 2024. URL
 676 <https://arxiv.org/abs/2408.06292>.
 677

678 Ziming Luo, Zonglin Yang, Zexin Xu, Wei Yang, and Xinya Du. Llm4sr: A survey on large language
 679 models for scientific research. *arXiv:2501.04306*, 2025. URL <https://arxiv.org/abs/2501.04306>.
 680

681 Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models
 682 of self-supervised sequence models. In Yonatan Belinkov, Sophie Hao, Jaap Jumelet, Najoung
 683 Kim, Arya McCarthy, and Hosein Mohebbi (eds.), *Proceedings of the 6th BlackboxNLP Workshop:
 684 Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, Singapore, December 2023.
 685 Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.2. URL <https://aclanthology.org/2023.blackboxnlp-1.2>.
 686

687 OpenAI. Gpt-5 system card. <https://cdn.openai.com/gpt-5-system-card.pdf>,
 688 2025. Accessed: 2025-09-22.

689 Harshith Padigela, Chintan Shah, and Dinkar Juyal. Ml-dev-bench: Comparative analysis of ai agents
 690 on ml development workflows. *arXiv:2502.00964*, 2025. URL <https://arxiv.org/abs/2502.00964>.
 691

692 Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2code: Automating code
 693 generation from scientific papers in machine learning. *arXiv:2504.17192*, 2025. URL <https://arxiv.org/abs/2504.17192>.
 694

695 Chan Jun Shern, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio
 696 Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine
 697 learning agents on machine learning engineering. *arXiv:2410.07095*, 2024. URL <https://arxiv.org/abs/2410.07095>.
 698

699 Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can LLMs generate novel research ideas? a
 700 large-scale human study with 100+ NLP researchers. *arXiv:2409.04109*, 2024. URL <https://arxiv.org/abs/2409.04109>.
 701

702 Zachary S Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebel, and Arvind Narayanan. CORE-
 703 bench: Fostering the credibility of published research through a computational reproducibility
 704 agent benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL
 705 <https://openreview.net/forum?id=BsMMC4MEGS>.

706 Singularity. Singularity, 2021. URL <https://doi.org/10.5281/zenodo.1310023>.

707 Michael D. Skarlinski, Sam Cox, Jon M Laurent, James D. Braza, Michaela Hinks, Michael J.
 708 Hammerling, Manvitha Ponnappati, Samuel G. Rodrigues, and Andrew D. White. Language
 709 agents achieve superhuman synthesis of scientific knowledge. *arXiv:2409.13740*, 2024. URL
 710 <https://arxiv.org/abs/2409.13740>.

711 Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel
 712 Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: Evaluating ai's ability
 713 to replicate ai research. *arXiv:2504.01848*, 2025. URL <https://arxiv.org/abs/2504.01848>.

714 Zilu Tang, Mayank Agarwal, Alexander Shypula, Bailin Wang, Derry Wijaya, Jie Chen, and Yoon
 715 Kim. Explain-then-translate: an analysis on improving program translation with self-generated
 716 explanations. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for
 717 Computational Linguistics: EMNLP 2023*, pp. 1741–1788, Singapore, December 2023. Association
 718 for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.119. URL <https://aclanthology.org/2023.findings-emnlp.119/>.

719 The Terminal-Bench Team. Terminal-bench: A benchmark for ai agents in terminal environments,
 720 Apr 2025. URL <https://github.com/laude-institute/terminal-bench>.

721 Minyang Tian, Luyu Gao, Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan
 722 Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, HAO TONG, Kha Trinh,
 723 Chenyu Tian, Zihan Wang, Bohao Wu, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu,
 724 Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu A Huerta, and Hao Peng.
 725 Scicode: A research coding benchmark curated by scientists. In *The Thirty-eight Conference on
 726 Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL <https://openreview.net/forum?id=ADLaALtdoG>.

727 Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Yinxu Pan, Yesai Wu, Hui Haotian, Liu
 728 Weichuan, Zhiyuan Liu, and Maosong Sun. DebugBench: Evaluating debugging capability of
 729 large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the
 730 Association for Computational Linguistics: ACL 2024*, pp. 4173–4198, Bangkok, Thailand, August
 731 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.247. URL
 732 <https://aclanthology.org/2024.findings-acl.247/>.

733 Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,
 734 Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill
 735 Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan,
 736 Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers
 737 as generalist agents. In *The Thirteenth International Conference on Learning Representations*,
 738 2025. URL <https://openreview.net/forum?id=OJd3ayDDoF>.

739 Leon Weber-Genzel, Siyao Peng, Marie-Catherine De Marneffe, and Barbara Plank. VariErr NLI:
 740 Separating annotation error from human label variation. In Lun-Wei Ku, Andre Martins, and
 741 Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Com-
 742 putational Linguistics (Volume 1: Long Papers)*, pp. 2256–2269, Bangkok, Thailand, August
 743 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.123. URL
 744 <https://aclanthology.org/2024.acl-long.123/>.

745 Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim,
 746 Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limi-
 747 tations of language models through counterfactual tasks. In Kevin Duh, Helena Gomez, and
 748 Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of
 749 the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long
 750 751 752 753 754 755*

756 *Papers*), pp. 1819–1862, Mexico City, Mexico, June 2024. Association for Computational Linguistics.
 757 doi: 10.18653/v1/2024.nacl-long.102. URL <https://aclanthology.org/2024-naacl-long.102/>.

759
 760 Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. Scireplicate-bench: Bench-
 761 marking llms in agent-driven algorithmic reproduction from research papers. *arXiv:2504.00255*,
 762 2025. URL <https://arxiv.org/abs/2504.00255>.

763 Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, Jian-Guang Lou,
 764 and Shuai Ma. Re-reading improves reasoning in large language models. In Yaser Al-Onaizan,
 765 Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical
 766 Methods in Natural Language Processing*, pp. 15549–15575, Miami, Florida, USA, November
 767 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.871. URL
 768 <https://aclanthology.org/2024.emnlp-main.871/>.

769 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik
 770 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models.
 771 In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Ad-
 772 vances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Asso-
 773 ciates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

774 Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu
 775 Hu, Jie Tang, and Yisong Yue. Datascibench: An LLM agent benchmark for data science.
 776 *arXiv:2502.13897*, 2025. URL <https://arxiv.org/abs/2502.13897>.

777 Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. Can large
 778 language models transform computational social science? *Computational Linguistics*, 50(1):
 779 237–291, 03 2024. ISSN 0891-2017. doi: 10.1162/coli_a_00502. URL https://doi.org/10.1162/coli_a_00502.

780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809

810 A LIST OF PAPERS FOR REXBENCH EXTENSION
811
812
813814 Table 1: List of papers that form the bases for extensions in **REXBENCH**.
815

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

837
838 Table 1 shows the papers forming the basis of the RExBench extensions.
839
840
841842 B DETAILED EXPERIMENTAL SETUP
843
844845 Table 2: Resource requirements for each task.
846

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

862 Table 2 shows the details about the execution environment for each task.
863

864 C AN EXAMPLE TASK INSTRUCTION (EXTENSION OF WINODICT)
865866 **WinoDict Task Instruction**
867868 **PROBLEM DESCRIPTION**
869870 **BACKGROUND**
871

872 The paper *WinoDict: Probing language models for in-context word acquisition* (Eisenschlos et al. 2023)
 873 attempts to measure LLMs' ability to learn novel words during inference. They rewrite Winograd-style
 874 co-reference resolution problems by replacing the key concept word with a synthetic but plausible
 875 English word and adding the definition of the new concept as a suffix. Building on this work, we would
 876 like to further consider a learning setting where the form of the learned words coincides with existing
 877 English words and explore how their existing meanings may interfere with the models' word acquisition
 878 from the given definition. The hypothesis is that overriding existing words would be more difficult, and
 879 the frequency of the existing words may also modulate the effect.

880 The paper will be available inside the provided repository in both PDF format as
 881 `eisenschlos_et_al_2023.pdf` and markdown format as `eisenschlos_et_al_2023.md`
 882 if you need to refer to it.

883 **EXTENSION TO BE IMPLEMENTED**
884

885 Your task is to modify the codebase provided to generate new Winodict datasets by replacing the target
 886 word being learned with an existing English word. The new dataset should be stored under the directory
 887 `./data`. Your replacement should consider the POS tags of the original word - they should be matched.
 888 We will only consider four POS categories for word replacement: nouns, verbs, adjectives, and adverbs.
 889 To test the possible effect of frequency, sample the candidates from different frequency groups:

890 1. Top Group:
891

- 892 • Verbs, Nouns, Adverbs: Select the top 20% most frequent words
- 893 • Adjectives: Select the top 35% most frequent adjectives (to match the sample set size)

894 2. Bottom Group:
895

- 896 • Verbs, Nouns, Adverbs: Select the bottom 20% least frequent words
- 897 • Adjectives: Select the bottom 35% least frequent adjectives.

898 3. All Group:
899

- 900 • Verbs, Nouns, Adjectives, Adverbs: Include all words, no frequency-based filtering

901 Assume that the frequency information will be provided in a form of four files corresponding
 902 to each POS, named `1_all_rank_noun.txt`, `2_all_rank_verb.txt`,
 903 `3_all_rank_adjective.txt`, `4_all_rank_adverb.txt`, under the directory `./words/`.
 904 Each file lists words in descending order of frequency from the British National Corpus. To generate the
 905 dataset, you need to create word candidates based on the files and sample words from those candidates.

906 From each group, sample words from the candidate lists to generate the new Winodict dataset. Ensure
 907 that the replacement word is inflected to match the morphological properties of the original word being
 908 replaced. For instance, if the original word is a past tense verb, the selected replacement must also be
 909 in the past tense. Please use spaCy with the `lemminflect` module to inflect the selected words as
 910 necessary.

911 Using the new dataset, you should run experiments on the Winodict-Winograd dataset under the 5-shot
 912 setting. Assume that the model can be found under `/stage/hf_cache/gemma-2-9b`, although
 913 this may not be visible in the provided repository right now. Furthermore, we will only consider the
 914 setting where definitions are appended as suffixes, which are represented as the `last_def` template in
 915 the codebase. Save your results as three different files under `./results/`, corresponding to the three
 916 sampling groups defined above. They should be named `res_top.json`, `res_bottom.json`, and
 917 `res_all.json`.

918 Please make the experiment runnable by implementing a single script called `run_final.sh` in the
 919 root of the repository. This script should call all necessary commands with all parameters specified and
 920 should not have any command line arguments itself. It should handle both the dataset generation as well
 921 as the execution of the experiments on this new dataset.

922 Try your best to keep everything else not specified above constant in the original repository. Also, the
 923 environment is already set up, so you do not need to install any dependencies or download any more
 924 datasets/models. Please refer to `environment.yml` in the repository for an overview of the installed
 925 libraries and their versions.

926 **EVALUATION**
927

928 To evaluate the extension, we will execute the `run_final.sh` script you wrote. We will use the three
 929 `.json` files mentioned above that contain the final results to evaluate the experimental outcome.

918 **D DETAILED AGENT CONFIGURATIONS**
919920 **Table 3: Agent components**
921

922

Component	aider	Claude Code	OpenHands
Repo navigation	✗	✓	✓
Tool use	✗	✓	✓
Bash execution	✗	✓	✓
Python execution	✗	✓	✗

922 Table 3 provides an overview of what kind of abilities each agent has.
923924 **aider** aider is an open-source agent framework. We implemented our most basic agent based on
925 aider, using the “diff” edit format where the LLM specifies file changes as search/replace blocks.
926 We allowed up to 5 retries to handle API-side overload errors. Since aider lacks built-in file search
927 capabilities, we added a preliminary stage where the LLM is given the codebase’s directory tree along
928 with the task instruction to identify files requiring modification. Unlike Claude Code and OpenHands,
929 our aider implementation does not use bash execution or tools.
930931 **Claude Code** Claude Code is an interactive command-line tool that can be run from within a
932 repository. We used Claude 3.7 Sonnet as the backbone in our experiments. Claude Code can then
933 navigate the repository and make edits to files or create new files. We manually evaluated Claude
934 Code with the prompt “Read the instructions in instructions.md and carry out the specified task.”
935 Furthermore, to avoid Python executions by the agent, we added the instruction “Please do not
936 execute any code, just read relevant files and make any necessary modifications.” to the task-specific
937 instructions. Since this tool does not allow for modification of system prompts or support any other
938 customization, everything else about this agent was left as defined by its developers. We additionally
939 applied minimal postprocessing to patches (only needed for the Re-reading task) containing an
940 absolute filepath—the agent was evaluated locally and receives the absolute path to the codebase
941 directory as input in an uncontrollable fashion, but patches are evaluated inside a virtual machine
942 with a different filepath structure.
943944 **OpenHands** OpenHands is an open-source agent framework that uses an LLM to control a range
945 of pre-defined tools for understanding and modifying codebases. To make this agent more fairly
946 comparable to Claude Code, we modified the system prompt and the agent to disable execution of
947 Python code. The agent was allowed to execute bash commands such as `grep` and `cat`, browse
948 the web, load PDFs in a browser (if compatible with the backbone LLM), and edit files. We
949 prompted this agent with the same one-line instruction as for Claude Code. We evaluated this agent
950 in “headless” mode in which the agent executes the task without any user input until the LLM signals
951 task completion to the agent, or the agent detects a loop or reaches a maximum number of steps (250).
952 As with Claude Code, we applied postprocessing to absolute filepaths to make them compatible with
953 the virtual machine evaluation environment, since the OpenHands agent is run inside its own Docker
954 container.
955956 **E TOOL USE/ACTION DISTRIBUTION**
957958 OpenHands agents interact with external tools during execution, and we analyze how their tool usage
959 varies across different LLMs. Claude 4 Sonnet and OpenAI GPT-5 showed the highest overall tool
960 usage. File operations (`str_replace_editor`) and bash commands (`execute_bash`) were
961 the most frequently used tools across all models (see Figure 6) but occasionally the agent did also
962 perform web searches or use a browser to render the paper PDF.
963964 **F DETAILED EXPERIMENTAL RESULTS**
965966 Table 4 shows the detailed results for all metrics and each agent-LLM combination across all three
967 hint levels.
968

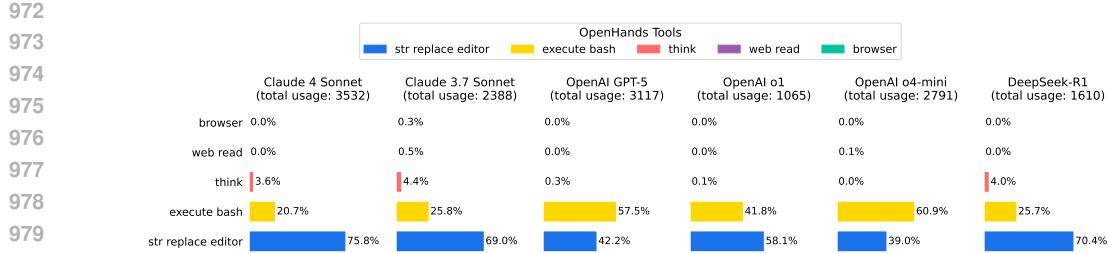


Figure 6: Tool usage distribution across OpenHands agent implementations. Percentages indicate the frequency of each tool type, while the total usage count is shown in each column header.

Table 5 shows the number of turns as well as the number of input and output tokens, averaged across the three runs for each agent.

Table 6 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 7 and Table 8 shows the detailed breakdown of errors for each agent and LLM combination.

Tables 9 to 21 show the detailed breakdown of task specific performance for each agent and LLM combination.

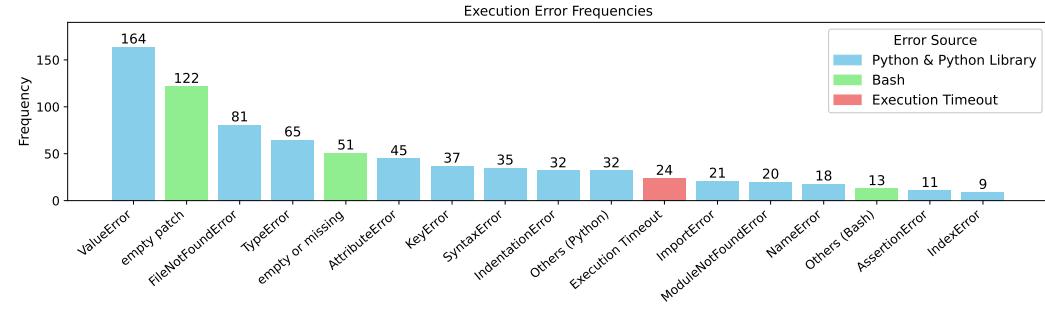


Figure 7: Distribution of execution errors across Python, Bash, and timeout categories. Errors with fewer than 5 occurrences are grouped as ‘Others’.

G 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 occasionally 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

1026 Table 4: Detailed performance on **REXBENCH**, evaluated across three hint levels. Results are
1027 averaged across three runs.
1028

| 1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045 |
|--|--|--|--|--|--|--|
| aider | Claude 4 Sonnet | No hints | 0.86 | 0.50 | 0.19 | |
| | | Hints | 0.84 | 0.47 | 0.31 | |
| | | Detailed Hints | 0.86 | 0.39 | 0.25 | |
| | Claude 3.7 Sonnet | No hints | 0.87 | 0.39 | 0.14 | |
| | | Hints | 0.86 | 0.31 | 0.17 | |
| | | Detailed Hints | 0.86 | 0.33 | 0.08 | |
| | OpenAI GPT-5 | No hints | 0.81 | 0.36 | 0.06 | |
| | | Hints | 0.84 | 0.42 | 0.06 | |
| | | Detailed Hints | 0.84 | 0.25 | 0.14 | |
| | OpenAI o1 | No hints | 0.84 | 0.22 | 0.00 | |
| | | Hints | 0.78 | 0.31 | 0.00 | |
| | | Detailed Hints | 0.80 | 0.39 | 0.03 | |
| | OpenAI o4-mini | No hints | 0.43 | 0.19 | 0.03 | |
| | | Hints | 0.43 | 0.25 | 0.19 | |
| | | Detailed Hints | 0.43 | 0.31 | 0.14 | |
| | DeepSeek-R1 | No hints | 0.18 | 0.00 | 0.00 | |
| | | Hints | 0.13 | 0.00 | 0.00 | |
| | | Detailed Hints | 0.13 | 0.00 | 0.00 | |
| Claude Code | Claude 3.7 Sonnet | No hints | 0.76 | 0.33 | 0.25 | |
| | | Hints | 0.84 | 0.50 | 0.28 | |
| | | Detailed Hints | 0.88 | 0.42 | 0.25 | |
| OpenHands | Claude 4 Sonnet | No hints | 0.79 | 0.67 | 0.31 | |
| | | Hints | 0.85 | 0.67 | 0.44 | |
| | | Detailed Hints | 0.90 | 0.78 | 0.47 | |
| | Claude 3.7 Sonnet | No hints | 0.76 | 0.42 | 0.25 | |
| | | Hints | 0.87 | 0.53 | 0.39 | |
| | | Detailed Hints | 0.92 | 0.53 | 0.36 | |
| | OpenAI GPT-5 | No hints | 0.79 | 0.72 | 0.31 | |
| | | Hints | 0.88 | 0.61 | 0.39 | |
| | | Detailed Hints | 0.87 | 0.69 | 0.42 | |
| | OpenAI o1 | No hints | 0.64 | 0.31 | 0.00 | |
| | | Hints | 0.67 | 0.33 | 0.08 | |
| | | Detailed Hints | 0.78 | 0.39 | 0.03 | |
| | OpenAI o4-mini | No hints | 0.68 | 0.39 | 0.08 | |
| | | Hints | 0.77 | 0.36 | 0.19 | |
| | | Detailed Hints | 0.74 | 0.47 | 0.14 | |
| | DeepSeek-R1 | No hints | 0.74 | 0.14 | 0.00 | |
| | | Hints | 0.76 | 0.14 | 0.00 | |
| | | Detailed Hints | 0.71 | 0.22 | 0.08 | |

1066 of the input in one of the experimental settings. Similarly, in the VariErr-NLI task OpenHands +
1067 GPT-5 unnecessarily modified an output file path required for obtaining the final scores, resulting in
1068 the evaluation scripts being unable to access the results of the agent’s implementation. Given that
1069 scientific work relies on rigor and reproducibility, deviations like these from the specified instructions
1070 are problematic. This highlights the need to design agents which conform exactly to the requirements
1071 given, without introducing additional unrequested changes.
1072

1073 H LICENSE INFORMATION

1076 The codebase portion of **REXBENCH** is constructed from public repositories—details of the licenses
1077 for each task are provided in Table 22. When the codebase did not contain any license information, we
1078 reached out to the authors for more information and used their suggestion (one response still pending
1079 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,

Table 5: Token usage statistics across agents and models.

Agent	Model	Hints Level	Total Turns (Avg.)	Prompt Tokens (Avg.)	Output Tokens (Avg.)
aider	Claude 4 Sonnet	No hints	2.00	3121.10	3694.40
		Hints	2.00	3055.70	3794.00
		Detailed Hints	2.00	3303.80	4254.70
	Claude 3.7 Sonnet	No hints	2.00	3053.60	5204.20
		Hints	2.00	3029.10	4222.20
		Detailed Hints	2.00	3529.20	3996.20
	OpenAI GPT-5	No hints	2.00	3280.60	8995.60
		Hints	2.00	3221.10	8544.10
		Detailed Hints	2.00	3466.40	9697.80
OpenAI o1	OpenAI o1	No hints	2.00	2964.60	5302.20
		Hints	2.00	3061.50	6052.80
		Detailed Hints	2.00	3447.60	6011.10
OpenAI o4-mini	OpenAI o4-mini	No hints	2.00	2910.60	4286.10
		Hints	2.00	3002.80	2875.30
		Detailed Hints	2.00	4457.30	3388.80
DeepSeek-R1	DeepSeek-R1	No hints	2.00	2963.80	3557.10
		Hints	2.00	3045.80	3751.50
		Detailed Hints	2.00	3446.20	3378.70
Claude Code	Claude 3.7 Sonnet	No hints	34.64	–	–
		Hints	29.61	–	–
		Detailed Hints	25.00	–	–
OpenHands	Claude 4 Sonnet	No hints	94.53	1,455,087.28	10,080.81
		Hints	88.97	1,519,934.47	9370.17
		Detailed Hints	97.58	1,866,521.03	9738.08
	Claude 3.7 Sonnet	No hints	50.94	542,311.69	7492.75
		Hints	47.92	540,580.78	7116.36
		Detailed Hints	43.72	458,430.67	6882.17
	OpenAI GPT-5	No hints	67.25	973,985.28	34,009.06
		Hints	72.72	1,011,380.39	40,850.58
		Detailed Hints	70.44	882,550.53	37,924.22
OpenAI o1	OpenAI o1	No hints	16.58	82,144.81	9592.97
		Hints	23.61	137,508.86	13,816.28
		Detailed Hints	27.94	183,731.94	18,861.33
OpenAI o4-mini	OpenAI o4-mini	No hints	53.47	522,734.39	25,140.58
		Hints	54.64	511,121.25	25,899.81
		Detailed Hints	57.72	565,430.33	26,719.36
DeepSeek-R1	DeepSeek-R1	No hints	34.00	194,548.08	18,099.22
		Hints	34.78	202,214.92	19,673.08
		Detailed Hints	36.19	293,836.28	20,307.64

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.

1134
1135

Table 6: Cost and duration statistics across agents and models (main experiment).

1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148

Agent	Model	Avg. Cost (\$)	Avg. Duration	Total Cost (\$)	Total Duration
aider	Claude 4 Sonnet	0.19	1m 4s	6.99	38m 27s
	Claude 3.7 Sonnet	0.41	1m 44s	14.77	1h 2m 41s
	OpenAI GPT-5	0.10	2m 58s	3.61	1h 46m 53s
	OpenAI o1	0.62	3m 7s	22.37	1h 51m 59s
	OpenAI o4-mini	0.03	42s	1.02	24m 56s
	DeepSeek-R1	0.04	4m 38s	1.46	2h 42m 10s
Claude Code	Claude 3.7 Sonnet	0.60	2m 45s	21.94	1h 38m 45s
OpenHands	Claude 4 Sonnet	4.52	6m 17s	162.59	3h 46m 16s
	Claude 3.7 Sonnet	0.40	2m 43s	14.22	1h 38m 6s
	OpenAI GPT-5	1.14	12m 52s	41.20	7h 43m 14s
	OpenAI o1	1.30	2m 25s	46.93	1h 27m 4s
	OpenAI o4-mini	0.61	5m 2s	22.09	3h 1m 45s
	DeepSeek-v3	0.03	4m 41s	1.01	2h 49m 5s

1149
1150

Table 7: Breakdown of error counts for Aider and Claude Code.

1151
1152

Error Type	aider						Claude Code
	Claude 4 Sonnet	Claude 3.7 Sonnet	OpenAI GPT-5	OpenAI o1	OpenAI o4-mini	DeepSeek R1	
Python Errors							
AssertionError	1	0	4	0	1	0	0
AttributeError	3	11	4	5	2	0	3
FileNotFoundException	14	23	7	11	4	2	6
ImportError	1	6	2	6	0	0	2
IndentationError	0	0	0	0	0	0	0
IndexError	0	0	1	1	0	0	1
IsADirectoryError	0	0	0	0	0	0	0
KeyError	0	5	1	2	6	1	4
LookupError	0	0	1	0	0	0	0
ModuleNotFoundError	2	0	5	3	0	0	0
NameError	1	1	1	1	0	1	0
NotImplementedError	0	0	0	0	0	0	0
OSError	0	0	0	0	0	0	0
RuntimeError	0	0	1	1	0	0	0
SyntaxError	1	0	0	5	4	0	2
TypeError	1	3	4	2	6	3	10
UnboundLocalError	1	0	0	1	0	0	0
ValueError	28	10	30	8	6	4	24
EOFError	0	1	0	0	0	0	0
Python Library Errors							
DatasetNotFoundError	1	1	0	0	0	0	0
NotFoundError	2	0	1	1	0	0	1
OutOfMemory	0	0	0	0	0	0	0
ArgumentError	0	0	0	0	0	0	0
ScannerError	0	0	0	0	0	0	1
Other Python Errors							
ConstructorError	1	0	0	0	0	0	0
JSONDecodeError	2	0	3	0	0	0	0
Bash Errors							
cannot create directory	0	0	0	1	0	0	0
empty patch	0	0	0	0	35	59	0
empty or missing	0	0	2	4	5	25	0
unable to write file	0	0	0	0	0	3	0
Permission denied	0	0	0	2	0	0	0
syntax error	0	0	0	0	0	0	1
cannot access	0	0	0	0	0	0	0
Execution Timeout	0	2	0	7	1	1	4

1188

1189

1190

1191

1192

1193

1194

1195

1196

Table 8: Breakdown of error counts for OpenHands.

1197

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	2	2	0
AttributeError	0	7	1	2	2	5
FileNotFoundException	3	4	0	3	1	3
ImportError	0	0	0	0	1	3
IndentationError	0	0	3	10	8	11
IndexError	3	1	0	1	0	1
IsADirectoryError	0	0	0	0	0	1
KeyError	1	4	1	1	2	10
LookupError	0	0	0	0	0	0
ModuleNotFoundError	4	3	1	0	1	1
NameError	0	0	2	5	2	4
NotImplementedError	0	0	0	0	1	0
OSError	0	0	0	0	0	1
RuntimeError	0	0	0	0	0	0
SyntaxError	0	0	0	13	7	5
TypeError	5	13	2	2	7	7
UnboundLocalError	0	0	0	0	0	0
ValueError	11	11	17	4	3	8
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	0	0	2	0
ArgumentError	0	0	0	0	5	0
ScannerError	0	0	0	0	0	0
Other Python Errors						
ConstructorError	0	0	0	0	0	0
JSONDecodeError	0	0	0	0	0	0
Bash Errors						
cannot create directory	0	0	0	1	0	0
empty patch	0	0	0	9	10	9
empty or missing	0	1	0	3	7	4
unable to write file	0	0	0	0	0	0
Permission denied	0	0	0	1	0	1
syntax error	3	0	0	0	0	0
cannot access	0	1	0	0	0	0
Execution Timeout	0	0	2	4	2	1

1234

1235

1236

1237

1238

1239

1240

1241

1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251

Table 9: Detailed performance on aider + Claude 4 Sonnet.

1253	Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1254	aider	Claude 4 Sonnet	No Hints	CheckEval	1.00	1.00	0.00
1255				COGS	0.50	0.00	0.00
1256				Entity Tracking	1.00	0.67	0.33
1257				Explain then Translate	1.00	1.00	0.00
1258				Instruction Tuning	0.50	0.00	0.00
1259				Mission Impossible	1.00	1.00	0.00
1260				Othello	1.00	1.00	1.00
1261				Reasoning or Reciting	1.00	0.00	0.00
1262				Re-reading	0.52	0.33	0.00
1263				Tree of Thoughts	1.00	0.00	0.00
1264				VariErr-NLI	1.00	0.00	0.00
1265				WinoDict	0.75	1.00	1.00
1266			Hints	CheckEval	1.00	1.00	0.00
1267				COGS	0.50	0.00	0.00
1268				Entity Tracking	1.00	1.00	1.00
1269				Explain then Translate	0.80	0.00	0.00
1270				Instruction Tuning	0.50	0.00	0.00
1271				Mission Impossible	1.00	1.00	0.67
1272				Othello	1.00	1.00	1.00
1273				Reasoning or Reciting	1.00	0.33	0.00
1274				Re-reading	0.60	0.33	0.00
1275				Tree of Thoughts	1.00	0.00	0.00
1276			Detailed Hints	VariErr-NLI	0.90	0.00	0.00
1277				WinoDict	0.75	1.00	1.00
1278				CheckEval	1.00	0.33	0.00
1279				COGS	1.00	0.00	0.00
1280				Entity Tracking	1.00	0.33	0.00
1281				Explain then Translate	1.00	1.00	1.00
1282				Instruction Tuning	0.40	0.00	0.00
1283				Mission Impossible	1.00	0.67	0.33
1284				Othello	1.00	1.00	0.67
1285				Reasoning or Reciting	1.00	0.00	0.00
1286				Re-reading	0.60	0.33	0.00
1287				Tree of Thoughts	0.60	0.00	0.00
1288				VariErr-NLI	1.00	0.00	0.00
1289				WinoDict	0.75	1.00	1.00

1290
 1291
 1292
 1293
 1294
 1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

Table 10: Detailed performance on aider + Claude 3.7 Sonnet.

1306

1307 Agent	1308 Model	1309 Hint Level	1310 Task	1311 File Recall	1312 Execution Success	1313 Final Success
aider	Claude 3.7 Sonnet	No Hints	CheckEval	1.00	0.00	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.33	0.33
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.67	0.00
			Mission Impossible	1.00	1.00	0.33
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	0.60	0.00	0.00
			Re-reading	1.00	0.33	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.00	0.00
			COGS	0.50	0.00	0.00
- Hints		Detailed Hints	Entity Tracking	1.00	0.67	0.33
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.67
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.00	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	1.00
1340		Detailed Hints	Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
1341		Detailed Hints	Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
1342		Detailed Hints	CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
1343		Detailed Hints	Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
1344		Detailed Hints	Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
1345		Detailed Hints	Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
1346		Detailed Hints	CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
1347		Detailed Hints	Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
			Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00
			Mission Impossible	1.00	1.00	0.00
1348		Detailed Hints	Othello	1.00	1.00	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.33	0.00
			CheckEval	1.00	0.67	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.50	0.00	0.00

1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358

Table 11: Detailed performance on aider + GPT-5.

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
aider	GPT-5	No Hints	CheckEval	0.80	0.33	0.00
			COGS	1.00	0.33	0.00
			Entity Tracking	1.00	0.33	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	1.00	0.00	0.00
			Mission Impossible	1.00	0.33	0.00
			Othello	1.00	0.67	0.00
			Reasoning or Reciting	0.20	0.33	0.00
			Re-reading	0.40	0.00	0.00
			Tree of Thoughts	0.90	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.60	1.00	0.67
		Hints	CheckEval	1.00	1.00	0.00
			COGS	1.00	0.33	0.00
			Entity Tracking	1.00	0.67	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	1.00	0.00	0.00
			Mission Impossible	1.00	0.33	0.00
			Othello	1.00	0.67	0.00
			Reasoning or Reciting	0.20	0.00	0.00
			Re-reading	0.56	0.33	0.00
			Tree of Thoughts	0.70	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.67	0.67
		Detailed Hints	CheckEval	1.00	0.00	0.00
			COGS	1.00	1.00	0.00
			Entity Tracking	0.80	0.00	0.00
			Explain then Translate	1.00	1.00	0.67
			Instruction Tuning	1.00	0.00	0.00
			Mission Impossible	1.00	0.33	0.00
			Othello	1.00	0.00	0.00
			Reasoning or Reciting	0.20	0.00	0.00
			Re-reading	0.48	0.00	0.00
			Tree of Thoughts	1.00	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	1.00	1.00

1397
 1398
 1399
 1400
 1401
 1402
 1403

Table 12: Detailed performance on aider + o1.

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1415 aider	1416 o1	1417 No Hints	1418 CheckEval	0.83	0.33	0.00
1419	1420	1421	1422 COGS	1.00	0.00	0.00
1423	1424	1425	1426 Entity Tracking	1.00	1.00	0.00
1427	1428	1429	1430 Explain then Translate	1.00	0.33	0.00
1431	1432	1433	1434 Instruction Tuning	0.00	0.00	0.00
1435	1436	1437	1438 Mission Impossible	1.00	0.33	0.00
1439	1440	1441	1442 Othello	1.00	0.00	0.00
1443	1444	1445	1446 Reasoning or Reciting	0.33	0.00	0.00
1447	1448	1449	1450 Re-reading	1.00	0.00	0.00
1451	1452	1453	1454 Tree of Thoughts	0.83	0.00	0.00
1455	1456	1457 VariErr-NLI	1.00	0.00	0.00	
1458	1459	1460 WinoDict	0.75	0.67	0.00	
1461	1462	1463	1464 Detailed Hints	0.67	0.00	0.00
1465	1466	1467	1468 CheckEval	0.67	0.00	0.00
1469	1470	1471	1472 COGS	1.00	0.67	0.00
1473	1474	1475	1476 Entity Tracking	1.00	1.00	0.00
1477	1478	1479	1480 Explain then Translate	1.00	0.00	0.00
1481	1482	1483	1484 Instruction Tuning	0.00	0.00	0.00
1485	1486	1487	1488 Mission Impossible	1.00	0.67	0.00
1489	1490	1491	1492 Othello	1.00	0.33	0.00
1493	1494	1495	1496 Reasoning or Reciting	0.60	0.00	0.00
1497	1498	1499	1500 Re-reading	1.00	0.00	0.00
1501	1502	1503	1504 Tree of Thoughts	1.00	1.00	0.00
1505	1506	1507	1508 VariErr-NLI	1.00	0.00	0.00
1509	1510	1511	1512 WinoDict	0.67	0.67	0.00

1458

1459

1460

1461

1462

1463

1464

1465

Table 13: Detailed performance on aider + o4-mini.

1466

1467

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1469 aider	1470 o4-mini	1471 No Hints	1472 CheckEval	1473 0.67	1474 0.00	1475 0.00
			1476 COGS	1477 0.00	1478 0.00	1479 0.00
			1480 Entity Tracking	1481 0.67	1482 0.67	1483 0.00
			1484 Explain then Translate	1485 0.67	1486 0.33	1487 0.00
			1488 Instruction Tuning	1489 0.00	1490 0.00	1491 0.00
			1492 Mission Impossible	1493 0.83	1494 0.33	1495 0.33
			1496 Othello	1497 1.00	1498 0.33	1499 0.00
			1500 Reasoning or Reciting	1501 0.27	1502 0.00	1503 0.00
			1504 Re-reading	1505 0.00	1506 0.00	1507 0.00
			1508 Tree of Thoughts	1509 0.83	1510 0.00	1511 0.00
			1512 VariErr-NLI	1513 0.33	1514 0.00	1515 0.00
			1516 WinoDict	1517 0.25	1518 0.33	1519 0.00
		1520 Hints	1521 CheckEval	1522 0.00	1523 0.00	1524 0.00
			1525 COGS	1526 0.00	1527 0.00	1528 0.00
			1529 Entity Tracking	1530 1.00	1531 1.00	1532 1.00
			1533 Explain then Translate	1534 1.00	1535 0.00	1536 0.00
			1537 Instruction Tuning	1538 0.00	1539 0.00	1540 0.00
			1541 Mission Impossible	1542 1.00	1543 0.67	1544 0.33
			1545 Othello	1546 1.00	1547 1.00	1548 1.00
			1549 Reasoning or Reciting	1550 0.33	1551 0.00	1552 0.00
			1553 Re-reading	1554 0.00	1555 0.00	1556 0.00
			1557 Tree of Thoughts	1558 0.17	1559 0.00	1560 0.00
			1561 VariErr-NLI	1562 0.33	1563 0.00	1564 0.00
			1565 WinoDict	1566 0.58	1567 0.33	1568 0.00
		1569 Detailed Hints	1570 CheckEval	1571 0.33	1572 0.00	1573 0.00
			1574 COGS	1575 0.33	1576 0.67	1577 0.00
			1578 Entity Tracking	1579 1.00	1580 1.00	1581 0.67
			1582 Explain then Translate	1583 0.67	1584 0.67	1585 0.33
			1586 Instruction Tuning	1587 0.17	1588 0.33	1589 0.00
			1590 Mission Impossible	1591 0.67	1592 0.33	1593 0.00
			1594 Othello	1595 0.67	1596 0.67	1597 0.67
			1599 Reasoning or Reciting	1600 0.47	1601 0.00	1602 0.00
			1604 Re-reading	1605 0.00	1606 0.00	1607 0.00
			1610 Tree of Thoughts	1611 0.67	1612 0.00	1613 0.00
			1616 VariErr-NLI	1617 0.67	1618 0.00	1619 0.00
			1622 WinoDict	1623 0.42	1624 0.00	1625 0.00

1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1520
 1521

Table 14: Detailed performance on aider + Deepseek-R1.

1522	Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1523	aider	Deepseek-R1	No Hints	CheckEval	0.00	0.00	0.00
1524				COGS	0.00	0.00	0.00
1525				Entity Tracking	0.00	0.00	0.00
1526				Explain then Translate	0.00	0.00	0.00
1527				Instruction Tuning	0.33	0.00	0.00
1528				Mission Impossible	0.67	0.00	0.00
1529				Othello	0.00	0.00	0.00
1530				Reasoning or Reciting	0.00	0.00	0.00
1531				Re-reading	0.00	0.00	0.00
1532				Tree of Thoughts	0.00	0.00	0.00
1533				VariErr-NLI	0.33	0.00	0.00
1534				WinoDict	0.00	0.00	0.00
1535			Hints	CheckEval	0.33	0.00	0.00
1536				COGS	0.00	0.00	0.00
1537				Entity Tracking	0.00	0.00	0.00
1538				Explain then Translate	0.00	0.00	0.00
1539				Instruction Tuning	0.00	0.00	0.00
1540				Mission Impossible	0.00	0.00	0.00
1541				Othello	0.00	0.00	0.00
1542				Reasoning or Reciting	0.00	0.00	0.00
1543				Re-reading	0.00	0.00	0.00
1544				Tree of Thoughts	0.00	0.00	0.00
1545				VariErr-NLI	0.33	0.00	0.00
1546			Detailed Hints	WinoDict	0.00	0.00	0.00
1547				CheckEval	0.00	0.00	0.00
1548				COGS	0.00	0.00	0.00
1549				Entity Tracking	0.33	0.00	0.00
1550				Explain then Translate	0.00	0.00	0.00
1551				Instruction Tuning	0.00	0.00	0.00
1552				Mission Impossible	0.00	0.00	0.00
1553				Othello	0.00	0.00	0.00
1554				Reasoning or Reciting	0.13	0.00	0.00
1555				Re-reading	0.00	0.00	0.00
1556				Tree of Thoughts	0.00	0.00	0.00
1557				VariErr-NLI	0.67	0.00	0.00
1558				WinoDict	0.00	0.00	0.00

1559
 1560
 1561
 1562
 1563
 1564
 1565

1566

1567

1568

1569

1570

1571

1572

1573

1574

1575

1576

Table 15: Detailed performance on Claude Code + Claude 3.7 Sonnet.

1577

1578	Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1579	Claude-Code	Claude 3.7 Sonnet	No Hints	CheckEval	0.50	0.00	0.00
1580				COGS	0.50	1.00	1.00
1581				Entity Tracking	1.00	0.00	0.00
1582				Explain then Translate	1.00	1.00	0.67
1583				Instruction Tuning	0.83	0.00	0.00
1584				Mission Impossible	0.67	0.33	0.00
1585				Othello	1.00	1.00	1.00
1586				Reasoning or Reciting	0.40	0.33	0.00
1587				Re-reading	1.00	0.00	0.00
1588				Tree of Thoughts	1.00	0.00	0.00
1589				VariErr-NLI	1.00	0.00	0.00
1590				WinoDict	0.25	0.33	0.33
1591	Hints	Claude 3.7 Sonnet	Hints	CheckEval	0.50	0.00	0.00
1592				COGS	0.50	1.00	1.00
1593				Entity Tracking	1.00	0.00	0.00
1594				Explain then Translate	1.00	1.00	0.33
1595				Instruction Tuning	0.83	0.33	0.33
1596				Mission Impossible	1.00	0.67	0.00
1597				Othello	1.00	1.00	1.00
1598				Reasoning or Reciting	0.40	0.00	0.00
1599				Re-reading	1.00	0.67	0.00
1600				Tree of Thoughts	1.00	1.00	0.67
1601	Detailed Hints	Claude 3.7 Sonnet	Detailed Hints	VariErr-NLI	1.00	0.67	0.00
1602				WinoDict	0.75	0.00	0.00
1603				CheckEval	0.83	0.00	0.00
1604				COGS	0.50	1.00	1.00
1605				Entity Tracking	1.00	0.33	0.33
1606				Explain then Translate	1.00	1.00	1.00
1607				Instruction Tuning	1.00	0.33	0.00
1608				Mission Impossible	1.00	1.00	0.67
1609				Othello	1.00	0.67	0.00
1610				Reasoning or Reciting	0.40	0.33	0.00
1611				Re-reading	1.00	0.33	0.00
1612				Tree of Thoughts	1.00	0.00	0.00
1613				VariErr-NLI	1.00	0.00	0.00
1614				WinoDict	0.75	0.33	0.00

1615

1616

1617

1618

1619

1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1630

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

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
OpenHands	Claude 4 Sonnet	No Hints	CheckEval	0.40	0.67	0.00
			COGS	0.60	1.00	1.00
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.67
			Instruction Tuning	1.00	0.33	0.33
			Mission Impossible	0.80	0.67	0.00
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	1.00	0.67	0.00
			Re-reading	0.40	0.67	0.00
			Tree of Thoughts	1.00	1.00	0.33
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.25	1.00	0.33
		Hints	CheckEval	0.60	0.00	0.00
			COGS	0.50	0.33	0.33
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	1.00	0.67	0.67
			Mission Impossible	1.00	1.00	0.67
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	1.00	0.67	0.00
			Re-reading	0.40	0.67	0.00
			Tree of Thoughts	1.00	1.00	0.33
		Detailed Hints	VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.65	0.67	0.33
			CheckEval	0.60	1.00	0.67
			COGS	1.00	0.67	0.67
			Entity Tracking	1.00	1.00	0.33
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	1.00	0.00	0.33
			Mission Impossible	1.00	1.00	0.67
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	1.00	1.00	0.00
		Detailed Hints	Re-reading	0.40	1.00	0.00
			Tree of Thoughts	1.00	1.00	0.67
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.75	0.67	0.33

1665
 1666
 1667
 1668
 1669
 1670
 1671
 1672
 1673

1674
 1675
 1676
 1677
 1678
 1679
 1680
 1681
 1682
 1683
 1684

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

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1686	OpenHands	Claude 3.7 Sonnet	No Hints	CheckEval	0.50	0.00
1687				COGS	0.50	1.00
1688				Entity Tracking	1.00	0.00
1689				Explain then Translate	1.00	1.00
1690				Instruction Tuning	1.00	0.67
1691				Mission Impossible	0.50	0.00
1692				Othello	1.00	1.00
1693				Reasoning or Reciting	0.40	0.00
1694				Re-reading	1.00	0.33
1695				Tree of Thoughts	1.00	0.33
1696				VariErr-NLI	1.00	0.33
1697				WinoDict	0.25	0.33
1698		Hints	CheckEval	0.67	0.67	0.33
1699			COGS	1.00	1.00	1.00
1700			Entity Tracking	1.00	1.00	1.00
1701			Explain then Translate	1.00	1.00	0.67
1702			Instruction Tuning	0.50	0.00	0.33
1703			Mission Impossible	1.00	0.67	0.00
1704			Othello	1.00	1.00	1.00
1705			Reasoning or Reciting	0.40	0.00	0.00
1706			Re-reading	1.00	0.00	0.00
1707			Tree of Thoughts	1.00	0.67	0.33
1708		Detailed Hints	VariErr-NLI	1.00	0.00	0.00
1709			WinoDict	0.75	0.00	0.00
1710			CheckEval	1.00	0.33	0.00
1711			COGS	1.00	1.00	1.00
1712			Entity Tracking	1.00	1.00	1.00
1713			Explain then Translate	1.00	1.00	1.00
1714			Instruction Tuning	1.00	0.67	0.00
1715			Mission Impossible	1.00	1.00	1.00
1716			Othello	1.00	1.00	0.00
1717			Reasoning or Reciting	0.40	0.00	0.00
1718			Re-reading	1.00	0.00	0.00
1719			Tree of Thoughts	0.83	0.33	0.33
1720			VariErr-NLI	1.00	0.00	0.00
1721			WinoDict	0.75	0.00	0.00

1722
 1723
 1724
 1725
 1726
 1727

1728

1729

1730

1731

1732

1733

1734

1735

1736

1737

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

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
OpenHands	GPT-5	No Hints	CheckEval	0.50	1.00	0.00
			COGS	0.50	0.67	0.67
			Entity Tracking	1.00	0.00	0.00
			Explain then Translate	1.00	1.00	0.67
			Instruction Tuning	1.00	0.00	0.00
			Mission Impossible	0.80	0.67	0.00
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	1.00	1.00	0.00
			Re-reading	0.40	1.00	0.00
			Tree of Thoughts	1.00	1.00	0.33
			VariErr-NLI	1.00	0.33	0.00
			WinoDict	0.30	1.00	1.00
		Hints	CheckEval	1.00	0.33	0.00
			COGS	0.50	1.00	1.00
			Entity Tracking	1.00	0.67	0.67
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.90	0.00	0.00
			Mission Impossible	1.00	0.00	0.00
			Othello	1.00	0.67	0.33
			Reasoning or Reciting	1.00	0.67	0.00
			Re-reading	0.40	1.00	0.00
			Tree of Thoughts	1.00	1.00	0.67
		Detailed Hints	VariErr-NLI	1.00	0.33	0.00
			WinoDict	0.75	0.67	0.67
			CheckEval	0.90	0.33	0.00
			COGS	0.50	1.00	1.00
			Entity Tracking	1.00	0.33	0.33
			Explain then Translate	1.00	1.00	1.00
			Instruction Tuning	0.90	0.00	0.00
			Mission Impossible	1.00	0.67	0.33
			Othello	1.00	1.00	1.00
			Reasoning or Reciting	1.00	1.00	0.00

1774

1775

1776

1777

1778

1779

1780

1781

1782
 1783
 1784
 1785
 1786
 1787
 1788
 1789
 1790
 1791

Table 19: Detailed performance on OpenHands + o1.

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
OpenHands	o1	No Hints	CheckEval	0.50	1.00	0.00
			COGS	0.33	0.00	0.00
			Entity Tracking	1.00	0.33	0.00
			Explain then Translate	1.00	1.00	0.00
			Instruction Tuning	0.17	0.00	0.00
			Mission Impossible	0.50	0.00	0.00
			Othello	1.00	0.00	0.00
			Reasoning or Reciting	0.20	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	0.83	0.67	0.00
			VariErr-NLI	0.67	0.00	0.00
			WinoDict	0.25	0.33	0.00
		Hints	CheckEval	0.50	1.00	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	1.00	1.00	1.00
			Explain then Translate	1.00	0.33	0.00
			Instruction Tuning	0.00	0.00	0.00
			Mission Impossible	0.50	0.67	0.00
			Othello	1.00	0.00	0.00
			Reasoning or Reciting	0.20	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	0.83	0.67	0.00
			VariErr-NLI	0.83	0.00	0.00
			WinoDict	0.17	0.33	0.00
		Detailed Hints	CheckEval	0.83	0.00	0.00
			COGS	0.83	1.00	0.00
			Entity Tracking	0.67	0.67	0.00
			Explain then Translate	1.00	0.67	0.33
			Instruction Tuning	1.00	0.33	0.00
			Mission Impossible	0.67	0.00	0.00
			Othello	1.00	0.33	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.00	0.00
			Tree of Thoughts	0.83	1.00	0.00
			VariErr-NLI	0.83	0.00	0.00
			WinoDict	0.25	0.33	0.00

1828
 1829
 1830
 1831
 1832
 1833
 1834
 1835

1836
 1837
 1838
 1839
 1840
 1841
 1842
 1843
 1844
 1845

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

1846	Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
1847	OpenHands	o4-mini	No Hints	CheckEval	0.50	0.33	0.00
1848				COGS	0.50	0.33	0.33
1849				Entity Tracking	1.00	1.00	0.33
1850				Explain then Translate	1.00	0.67	0.00
1851				Instruction Tuning	0.67	0.00	0.00
1852				Mission Impossible	0.17	0.00	0.00
1853				Othello	0.83	0.33	0.33
1854				Reasoning or Reciting	0.40	0.33	0.00
1855				Re-reading	1.00	0.67	0.00
1856				Tree of Thoughts	1.00	0.67	0.00
1857				VariErr-NLI	0.67	0.00	0.00
1858				WinoDict	0.25	0.33	0.33
1859			Hints	CheckEval	0.33	0.00	0.00
1860				COGS	0.50	0.67	0.67
1861				Entity Tracking	1.00	1.00	0.67
1862				Explain then Translate	1.00	0.33	0.00
1863				Instruction Tuning	0.67	0.00	0.00
1864				Mission Impossible	0.67	0.67	0.00
1865				Othello	1.00	0.67	0.67
1866				Reasoning or Reciting	0.27	0.00	0.00
1867				Re-reading	1.00	1.00	0.33
1868				Tree of Thoughts	1.00	0.00	0.00
1869				VariErr-NLI	1.00	0.00	0.00
1870			Detailed Hints	WinoDict	0.67	0.00	0.00
1871				CheckEval	0.50	0.33	0.00
1872				COGS	0.50	0.67	0.67
1873				Entity Tracking	0.67	0.67	0.00
1874				Explain then Translate	1.00	0.67	0.00
1875				Instruction Tuning	0.67	0.00	0.00
1876				Mission Impossible	1.00	0.67	0.00
1877				Othello	1.00	1.00	1.00
1878				Reasoning or Reciting	0.40	0.67	0.00
1879				Re-reading	1.00	1.00	0.00
1880				Tree of Thoughts	0.67	0.00	0.00
1881				VariErr-NLI	0.50	0.00	0.00
1882				WinoDict	0.67	0.00	0.00

1883
 1884
 1885
 1886
 1887
 1888
 1889

1890

1891 Table 21: Detailed performance on OpenHands + Deepseek-R1.

1892

1893

1894

Agent	Model	Hint Level	Task	File Recall	Execution Success	Final Success
OpenHands	Deepseek-R1	No Hints	CheckEval	0.50	0.00	0.00
			COGS	0.50	0.33	0.00
			Entity Tracking	0.67	0.00	0.00
			Explain then Translate	1.00	0.33	0.00
			Instruction Tuning	0.33	0.33	0.00
			Mission Impossible	0.33	0.00	0.00
			Othello	1.00	0.33	0.00
			Reasoning or Reciting	0.27	0.00	0.00
			Re-reading	0.67	0.00	0.00
			Tree of Thoughts	1.00	0.33	0.00
			VariErr-NLI	0.83	0.00	0.00
			WinoDict	0.25	0.00	0.00
		Hints	CheckEval	0.83	0.33	0.00
			COGS	0.50	0.00	0.00
			Entity Tracking	0.67	0.00	0.00
			Explain then Translate	1.00	0.67	0.00
			Instruction Tuning	0.67	0.00	0.00
			Mission Impossible	0.67	0.00	0.00
			Othello	1.00	0.33	0.00
			Reasoning or Reciting	0.40	0.00	0.00
			Re-reading	1.00	0.33	0.00
			Tree of Thoughts	0.83	0.00	0.00
			VariErr-NLI	1.00	0.33	0.00
			WinoDict	0.17	0.00	0.00
		Detailed Hints	CheckEval	1.00	0.00	0.00
			COGS	0.50	0.33	0.00
			Entity Tracking	0.67	0.67	0.00
			Explain then Translate	1.00	1.00	0.67
			Instruction Tuning	0.67	0.00	0.00
			Mission Impossible	1.00	0.00	0.00
			Othello	1.00	0.33	0.33
			Reasoning or Reciting	0.27	0.00	0.00
			Re-reading	1.00	0.33	0.00
			Tree of Thoughts	0.67	0.00	0.00
			VariErr-NLI	1.00	0.00	0.00
			WinoDict	0.25	0.33	0.00

1927

1928

1929

1930

Table 22: Licenses for each Github repository.

1931

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