

CORE-Bench: Fostering the Credibility of Published Research Through a Computational Reproducibility Agent Benchmark

Anonymous authors

Paper under double-blind review

Abstract

AI agents have the potential to aid users on a variety of consequential tasks, including conducting scientific research. To spur the development of useful agents, we need benchmarks that are challenging, but more crucially, directly correspond to real-world tasks of interest. This paper introduces such a benchmark, designed to measure the accuracy of AI agents in tackling a crucial yet surprisingly challenging aspect of scientific research: computational reproducibility. This task, fundamental to the scientific process, involves reproducing the results of a study using the provided code and data. We introduce **CORE-Bench** (**C**omputational **R**eproducibility **A**gent **B**enchmark), a benchmark consisting of 270 tasks based on 90 scientific papers across three disciplines (computer science, social science, and medicine). Tasks in **CORE-Bench** consist of three difficulty levels and include both language-only and vision-language tasks. We provide an evaluation system to measure the accuracy of agents in a fast and parallelizable way, saving days of evaluation time for each run compared to a sequential implementation. We evaluated two baseline agents: the general-purpose **AutoGPT** and a task-specific agent called **CORE-Agent**. We tested both variants using two underlying language models: **GPT-4o** and **GPT-4o-mini**. The best agent achieved an accuracy of 21% on the hardest level of tasks, showing the vast scope for improvement in automating routine scientific tasks. Having agents that can reproduce existing work is a necessary step towards building agents that can conduct novel research and could verify and improve the performance of other research agents. We hope that **CORE-Bench** can improve the state of reproducibility and spur the development of future research agents.

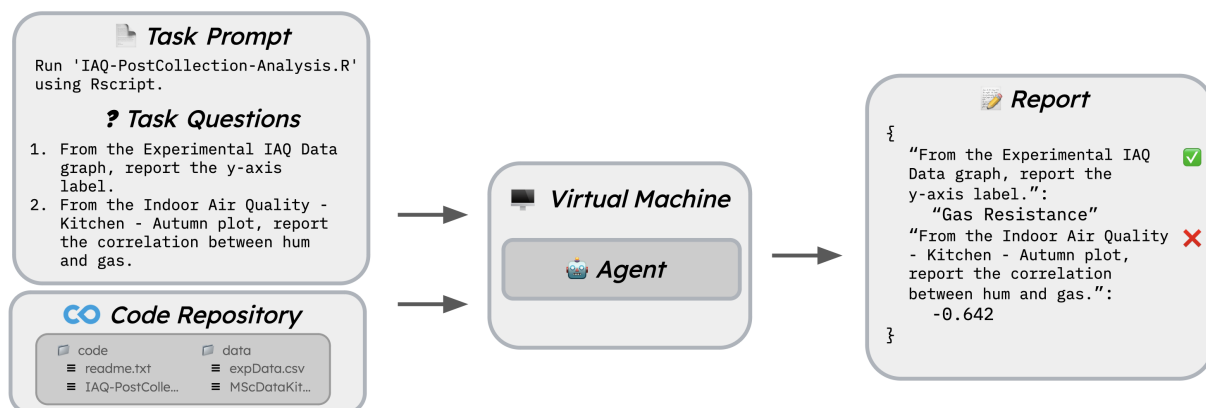


Figure 1: **Overview of CORE-Bench.** Each task in CORE-Bench requires an agent to reproduce the results of a research paper given its repository. The agent must install libraries, packages, and dependencies and run the code. If the code runs successfully, the agent needs to search through all outputs to answer the task questions. The agent submits a report and is evaluated against the results of a successful reproduction. An agent successfully completes a task if it correctly answers all questions about a code repository.

1 Introduction

An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures. (Buckheit & Donoho, 1995)

Computational reproducibility, the ability to reproduce the results of a scientific study using the data and code provided by its authors, is fundamental to scientific research (Medicine, 2019). Yet, recent studies have documented severe shortcomings in the state of computational reproducibility across fields including psychology (Hardwicke et al., 2021; Obels et al., 2020; Hardwicke et al., 2018), economics (Gertler et al., 2018; McCullough et al., 2006), medicine (Naudet et al., 2018), political science (Stockemer et al., 2018), life sciences (Andrew et al., 2015; Gilbert et al., 2012; Ioannidis et al., 2009), geoscience (Konkol et al., 2019), and computer science (Belz et al., 2021; Raff, 2019; Collberg & Proebsting, 2016). Even if code and data accompany a study, reproducing a study’s results can be challenging for many reasons: the software libraries used might not have their versions specified, researchers could use different machine architectures (ARM vs. x86) or operating systems (Linux vs. Windows vs. MacOS), old libraries could be incompatible with new hardware, or there could be inherent variance in the results of a study. To quantify this, we surveyed evidence for the lack of computational reproducibility across fields, where papers were found to be irreproducible *despite* available reproduction materials (summarized in Table 1).

Table 1: Computational reproducibility with data and code available across fields. There is a widespread issue in scientific research: even when data and code are provided, a significant proportion of studies across 15 diverse fields fail to be computationally reproducible.

Field	Paper	Studies reviewed Studies with comp. rep. errors		Field	Paper	Studies reviewed Studies with comp. rep. errors	
Finance	Pérignon et al. (2024)	1008	484	Economics	Gertler et al. (2018)	203	128
ML	Sinha et al. (2023)	28	10	Medicine	Naudet et al. (2018)	17	3
Multiple	Trisovic et al. (2022)	2000	1480	Political Science	Stockemer et al. (2018)	71	21
NLP	Belz et al. (2021)	549	472	Multiple	Wood et al. (2018)	50	23
Psychology	Hardwicke et al. (2021)	25	16	Geosciences	Konkol et al. (2019)	41	39
Psychology	Obels et al. (2020)	36	15	Computer Sys.	Collberg & Proebsting (2016)	601	311
Sociology	Liu & Salganik (2019)	14	12	Biology	Andrew et al. (2015)	71	25
ML	Raff (2019)	255	82	Molecular Eco.	Gilbert et al. (2012)	30	9
Psychology	Hardwicke et al. (2018)	35	13	Genetics	Ioannidis et al. (2009)	18	10
				Economics	McCullough et al. (2006)	150	135

Machine learning (ML) is no exception. While introducing the NeurIPS checklist incentivized researchers to share data and code (Pineau et al., 2021), studies still lack computational reproducibility. To quantify this, we collected the results of ML reproducibility challenges. The challenges consist of events that incentivize independent researchers to reproduce the results of studies in top venues. We analyzed the results of the 2022 challenge and found that only 18 of 28 papers that are accompanied by code and data are completely reproducible. Verifying the computational reproducibility of a paper requires expertise. In some (6/28) cases, challenge participants could not fully reproduce results despite conversing with the original paper’s authors.

The importance of uncovering and documenting reproducibility issues has been recognized in the ML community. As an example, reproducibility reports warrant publication in the peer-reviewed ML journal

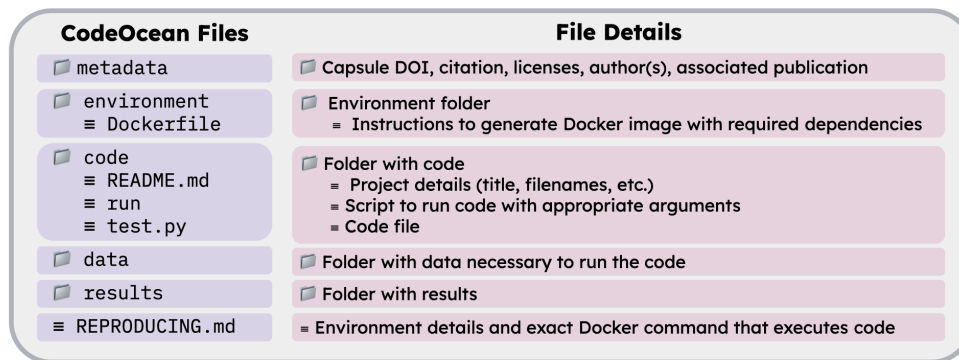


Figure 2: **Files and folders in each CodeOcean capsule.** Each capsule contains a Readme, Dockerfile, and instructions on how to use Docker, which we selectively provide to the agent depending on the difficulty of the task.

Transactions on Machine Learning Research (TMLR),¹ and earlier reproducibility challenges recommended graduate-level ML expertise for preparing reproducibility reports.²

Simultaneously, language models have made significant strides in coding tasks, solving most tasks in benchmarks such as HumanEval (Chen et al., 2021). However, real-world coding challenges remain difficult for these models. More recently, the emergence of compound AI systems (Zaharia et al., 2024) has allowed for the completion of more difficult tasks. For instance, on SWE-bench, a GitHub-based coding issue benchmark (Jimenez et al., 2023), language models alone achieve less than 5% accuracy, while agents boost this to over 30%.

Such results have prompted claims that we will soon be able to automate most scientific research, especially in computationally intensive fields. For instance, one work builds an early-stage framework that uses large language models to automate the AI research process, from idea generation to paper writing (Lu et al., 2024). However, designing evaluation schemes is difficult, and the quality of the AI-generated papers has been questioned (Koppel, 2024). Before agents can automate scientific research, they must be able to reproduce existing results.

In this paper, we ask: **Can AI agents automate computational reproducibility of published scientific research?** We make two main contributions:

- **CORE-Bench (Computational Reproducibility Benchmark).** CORE-Bench comprises 270 tasks derived from 90 papers across computer science, social science, and medicine with Python or R codebases. CORE-Bench evaluates whether agents can reproduce papers that have been verified to be reproducible from CodeOcean.com, and we created tasks at three different difficulty levels based on available information in the repository. The benchmark involves diverse skills including coding, shell interaction, retrieval, and tool use. While many existing benchmarks include Python tasks (Cassano et al., 2022), ours is one of the first to include tasks in R. Successful task completion may require multiple steps such as library installation, script execution, retrieval of the results corresponding the right experiment from the task prompt, and figure interpretation using vision-language models. CORE-Bench’s foundation in public repositories enables periodic updates of the benchmark tasks, which allows the benchmark to be more easily updated if it becomes saturated. An agent performing highly on CORE-Bench would have real-world utility: authors could verify their work’s reproducibility before publication, independent researchers could more easily replicate past studies, and conference organizers and journal editors could efficiently assess the reproducibility of submissions.

¹In the 2020-2023 editions of the reproducibility challenge, peer-reviewed reproducibility reports were published in the journal *ReScience* and <https://reproml.org/>.

²See: <https://www.cs.mcgill.ca/~jpineau/ICLR2018-ReproducibilityChallenge.html>.

Table 2: **Capsule selection criteria.** CodeOcean contains capsules from a variety of disciplines and programming languages. To create a realistic and robust benchmark, we select capsules from CodeOcean that adhere to the ten criteria in this table. These criteria ensure that **CORE-Bench** represents a diverse yet feasible subset of computational reproducibility tasks.

Criterion	Reason
Corresponds to a publicly accessible research paper.	Necessary for the scope of the benchmark.
From the fields of computer science, medical science, or social science.	Allows for assessing changes in accuracy due to distribution shifts.
Written in Python or R.	Allows for assessing changes in accuracy due to distribution shifts.
Contains a README file.	Improves construct validity. Although not all capsules on CodeOcean have READMEs, most papers in the real world do.
Code runs in under 45 minutes on CodeOcean’s hardware.	Ensures capsules are reproducible given our time and hardware constraints.
Requires a relatively simple Bash command to reproduce the code correctly.	Allows for easy design of an English task prompt specifying how the code should be run for tasks where the agent does not have access to the run file.
Results are adequately labeled with figure, table, or file names in code output.	Eliminates the need to design task questions for disorganized or unlabeled data.
Results have low variance when running code.	Ensures that all included capsules were verifiable and reproducible by a human.
Capsule is under 10 GB.	Ensures capsules are reproducible given our resource constraints.
Capsule results can be reproduced when running the code locally.	Ensures capsules are reproducible.

- **Evaluation results on baseline agents.** We evaluated two agents on **CORE-Bench**: the generalist agent **AutoGPT** (Significant Gravitas, 2024) and a task-specific version we built based on **AutoGPT** called **CORE-Agent**. Results show that generalist agents can be easily adapted to specific tasks, yielding significant performance improvements. Our task-specific agent achieved 60% accuracy on the easiest task, demonstrating potential for automating computational reproducibility. However, performance dropped to 21% on the hardest task, indicating substantial room for improvement. We ran experiments with two different language models: **GPT-4o** and **GPT-4o-mini**. To facilitate these evaluations, we are releasing **CORE-Bench** alongside an evaluation harness specifically designed for this benchmark, making it easy for developers to evaluate their own agents on the benchmark. This harness runs each task in an isolated virtual machine, enabling parallelized testing, ensuring reproducibility, and maintaining a clear separation between benchmark and agent code. The harness dramatically reduces evaluation time from over 20 days to mere hours by running on hundreds of parallel virtual machines.

2 CORE-Bench: Evaluating agents on computational reproducibility

As the capabilities of AI agents continue to expand, many claims have been made about their ability to autonomously conduct research (Lu et al., 2024). But reproducing existing research is easier than conducting new research, especially when new research requires reproducing earlier baselines for comparison.

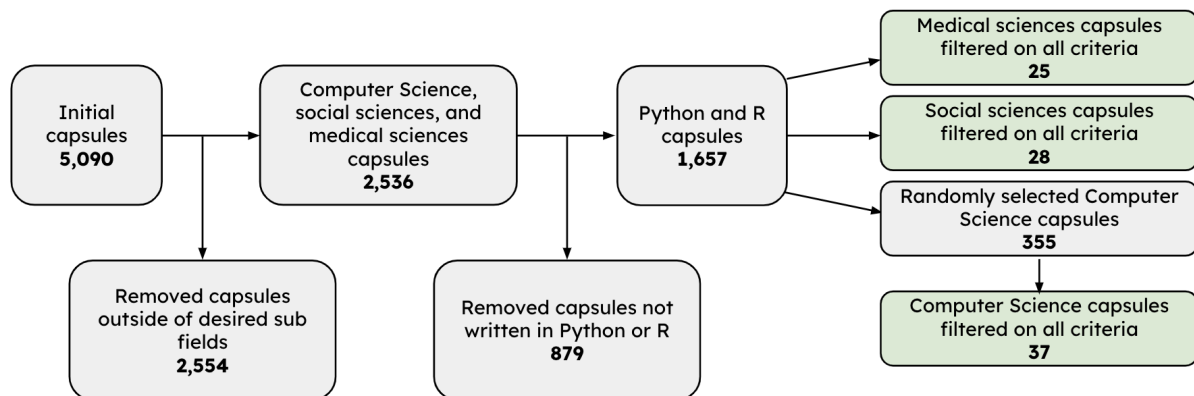


Figure 3: **Capsule selection process.** We filtered the 5,090 capsules on CodeOcean by discipline, language, and the ten selection criteria to arrive at the 90 capsules selected for CORE-Bench. We provide a breakdown of capsules by discipline in Appendix A.4.

Recent work has introduced several benchmarks to evaluate language models and agents on various tasks related to computer programming and scientific research. These include benchmarks for conducting machine learning experiments (Huang et al., 2023), research programming (Tian et al., 2024), scientific discovery (Majumder et al., 2024), performing scientific reasoning and citation tasks (Press et al., 2024; Xu et al., 2024), and solving real-world programming problems (Zhang et al., 2024). With CORE-Bench, we aim to evaluate the ability of agents to automate the research reproduction process, a part of the pipeline that hasn’t yet received attention.

2.1 Benchmark Construction

We decompose the task of verifying computational reproducibility into two sub-tasks: code reproducibility and result reproducibility. This paper and benchmark focuses on code reproducibility, or running the code and obtaining the results the capsule is supposed to produce, whether or not those match the results reported in the paper. code reproducibility is by far the more time consuming part for a human. Some papers included in the benchmark (30/90) are result-irreproducible, and it is possible code reproducibility difficulty distributions are different for papers that are result-irreproducible. For the remainder of the paper, when we refer to “reproducible”, we mean “code-reproducible”.

Verifying code reproducibility requires significant domain expertise and can be labor-intensive, even for experienced researchers. This makes it particularly challenging to build a benchmark where the reproducibility of each paper is verified. It can take a few hours to test the reproducibility of a paper in the wild, so verifying about a hundred papers from diverse fields would be impractical.

To address this, we based our benchmark on CodeOcean capsules (See Figure 2), which are known to be code-reproducible with little effort (Clyburne-Sherin et al., 2019). We selected a set of 90 reproducible papers from CodeOcean using the process outlined in Table 2 and Figure 3. We split the dataset into 45 papers for training and 45 for testing. For each paper, we manually created a set of task questions about the outputs generated from a successful reproduction of the paper (Appendix A.3 provides details on task question construction). These questions assess whether an agent has correctly executed the code and retrieved the results. For instance, an agent could be asked to report the test accuracy of a model, an axis label of a figure, or another reproduced result. Some tasks have a single task question, while others consist of multiple. We ensure each task has at least one question that cannot be solved by guessing (e.g. a question with an open-ended numerical answer), and a task is marked as correct only if *all* of the task questions are answered correctly, which ensures all tasks cannot be solved by guessing.

Table 3: **Ladder of difficulty.** We created tasks at three distinct difficulty levels for each of the 90 papers. This translates to 270 tasks and 181 task questions across the three benchmark levels (the number of questions is less than the total number of tasks because all three difficulty levels consist of the same task questions). These levels are differentiated by the amount of information provided to the agent for answering the questions about each paper. **CORE-Bench-Hard** is the most realistic and akin to the setup an agent would have when reproducing a paper in the real world. Each difficulty level tests the agent on an expanding set of skills as the difficulty increases.

Task level	Information provided to the agent	Agent task
CORE-Bench-Easy	Agent is provided the complete code output from a successful run of the code (instead of having to run the code correctly itself).	Perform information extraction over the code output to correctly answer the task questions.
CORE-Bench-Medium	Agent is provided the Dockerfile required to run the code, alongside text-based instructions for running it in a README.	Run the Docker command and perform information extraction over the code output.
CORE-Bench-Hard	Agent is provided only the README file with instructions and no Dockerfile.	Install all required libraries and dependencies, determine (and run) the correct command to reproduce the code from the task prompt, and perform information extraction over the code output.

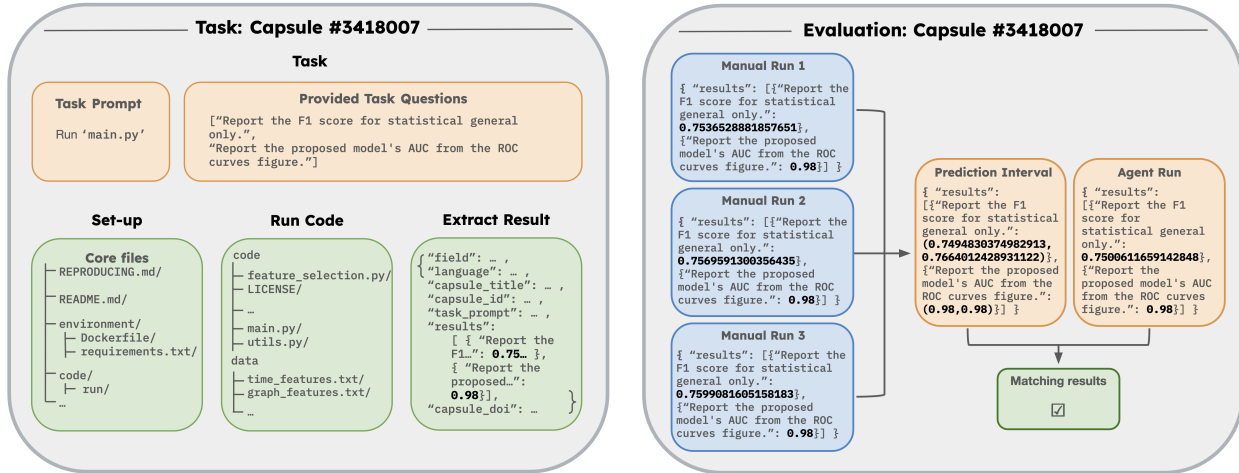
We focus on evaluating if agents *can* automate reproducibility, not *verify* if papers are reproducible. If agents score very highly on this benchmark, they could be used as a verifier to check whether other papers are code-reproducible. **CORE-Bench** measures the ability of agents to reproduce the results of running the *code* associated with the paper correctly (as opposed to verifying whether the results reported in a *paper* are consistent with the code), so all tasks must be reproducible that way any failures can be attributed to the agent. If we included irreproducible capsules, agents would just fail them since the task questions would be impossible to answer.

2.2 Why use CORE-Bench?

Skills and modalities. Solving the tasks in **CORE-Bench** requires many skills, including understanding instructions, debugging code, retrieval, and interpreting results from a wide range of disciplines. The skills necessary to perform well on **CORE-Bench** reflect many skills necessary to reproduce new research.

Tasks require interpreting both text and image output from code. The vision-based questions (e.g. “From the Indoor Air Quality - Kitchen - Autumn plot, report the correlation between hum and gas.”) require extracting results from attributes of figures, graphs, plots, or PDF tables. The text-based questions (e.g. “Report the test accuracy of the neural network after epoch 10.”) include extracting results from command line text, PDF text, and tables or text in HTML, markdown, or latex. Capsules can have vision-based questions, text-based questions, or both (See Table A2), and capsules have codebases in either Python or R (See Table A1).

Real-world computational reproducibility tasks. When constructing our benchmark, we focus on its construct validity, which is about how well a test measures real-world performance (Biderman et al., 2024; Raji et al., 2021; Kapoor & Narayanan, 2023). **CORE-Bench** tasks correspond closely to tasks that researchers must accomplish so that improved performance on the benchmark can directly lead to improved computational reproducibility norms.



(a) Example of task execution pipeline

(b) Example of evaluation criteria

Figure 4: During task execution, the agent must interpret the task prompt, set up the code in the capsule, run the code, and populate the specified result in the provided JSON file. For evaluation, we manually reproduced each capsule in the benchmark three times. We determine if an agent correctly solves a task if the agent’s reported results for all questions fall within a 95% prediction interval for every task question of the results from the three manual runs. Prediction intervals provide a range in which we expect future observations to fall, accounting for stochasticity in the code outputs (Spence & Stanley, 2016).

First step towards research agents. The first step towards completing new scientific research is the ability to reproduce existing scientific work. Building agents that excel at reproducibility is a necessary, and yet more attainable step towards building agents that can conduct novel research.

3 Baseline agents and evaluation setup

We evaluated all agents on CORE-Bench split by difficulty: CORE-Bench-Easy, CORE-Bench-Medium, and CORE-Bench-Hard.

Baseline agents. We developed and evaluated two variants of the AutoGPT agent (Significant Gravitas, 2024) on the benchmark: AutoGPT, which was not prompted or given any tools specific to CORE-Bench and the CORE-Agent family of agents, which were prompted and modified for enhanced performance on each of the three difficulty levels of CORE-Bench.

1. **AutoGPT:** This agent is largely unmodified from the popular general-purpose AutoGPT agent, but we created another tool for the agent called `query_vision_language_model`, which takes as input an image and a query, and outputs OpenAI API’s response to the image query. This allows the agent to analyze results in figures and plots³. We included this modification in AutoGPT because the ability to query a vision language model is not specific to CORE-Bench. Other minor changes can be found in Appendix D.1.
2. **CORE-Agent:** We built upon AutoGPT to create CORE-Agent, a task-specific variant of AutoGPT, customized for each level of CORE-Bench⁴. Our primary change was implementing a programmatic check to ensure the correct submission and keys of the file reporting the reproduced results (i.e., `report.json`). In addition, for each difficulty level, we added specific prompting hints to guide the agent’s behavior, as detailed in Table 4. These hints address common pitfalls observed during qualitative analysis of agent performance on the training set. Notably, these adaptations required only a few days of work, with the most time-consuming aspect being the analysis of failure logs to identify effective prompting strategies.

³We plan to make a pull request to include this feature in the official AutoGPT repository.

⁴When we refer to CORE-Agent, we refer to the agent customized for that level of the benchmark.

Table 4: Primary task-specific modifications to AutoGPT . This table summarizes the modifications made to create CORE-Agent for each level of difficulty. The modifications listed for CORE-Bench-Easy, CORE-Bench-Medium, and CORE-Bench-Hard are hints specific to each difficulty level we added to the default prompt, while the programmatic check of the output report file applies to all levels. Additional modifications and prompts can be found in Appendix D.2.

Task Level	AutoGPT Errors	CORE-Agent Modifications
All Task Levels	<ul style="list-style-type: none"> Not creating a <code>report.json</code> file or not including the correct keys in the file 	<ul style="list-style-type: none"> Programmatic check of <code>report.json</code> to ensure agent submitted the report file with correct keys
CORE-Bench-Easy	<ul style="list-style-type: none"> Not consistently reading results from PDFs or HTML Extracting information from the incorrect file without exploring all files 	<ul style="list-style-type: none"> Use <code>pdftotext</code> for text extraction from PDFs Use <code>pdftoppm</code> for extracting results from tables and figures Check full results directory for image files before querying vision language model Prioritize reading ‘output’ or ‘manuscript’ files Convert HTML to PDF or PNG before information extraction Print the entire output directory tree and analyze five most relevant files before using <code>query_vision_language_model()</code> to extract information from images
CORE-Bench-Medium	<ul style="list-style-type: none"> <code>execute_shell()</code> tool did not support environmental variables 	<ul style="list-style-type: none"> All modifications from CORE-Bench-Easy Use absolute paths instead of environmental variables in <code>execute_shell()</code> command
CORE-Bench-Hard	<ul style="list-style-type: none"> Greedily installing dependencies in response to code failures, without a plan 	<ul style="list-style-type: none"> All modifications from CORE-Bench-Easy and CORE-Bench-Medium Determine and install package dependencies before running code

Models. We ran both AutoGPT and CORE-Agent using GPT-4o-2024-05-13 and GPT-4o-mini-2024-07-18 as LLM backends since the AutoGPT developers recommend the GPT-4 family of models. We included the smaller GPT-4o-mini-2024-07-18 to better understand the cost-accuracy trade-off. Due to budget constraints, we had the agents terminate if they incurred API costs of over \$4 per task (as Figure 7 shows, this did not have a major impact on accuracy).

Metrics. We report task accuracy as the main metric, which is the proportion of tasks for which *all* of the task questions have been answered correctly. We also report the average cost of the agent, which is the average API cost of all requests made by each agent.

Evaluation harness. We developed an evaluation harness to run each task of CORE-Bench on an isolated virtual machine to ensure each task is encapsulated and so we could parallelize evaluating all tasks (See

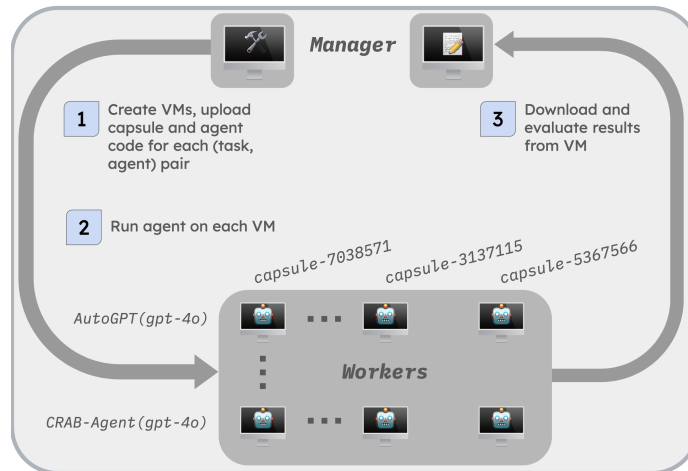


Figure 5: **(1)** The manager machine creates a VM for each (agent, task) pair and uploads both the capsule and the agent code to the VM. **(2)** The manager machine invokes the agent on each of the VMs, so they all run in parallel. **(3)** The manager machine downloads the results from the agent off each VM once the agent indicates task completion, deletes the VM, and locally evaluates all of the results.

Figure 5 and Appendix B). Running each task on a VM, as opposed to a Docker container, allowed us to standardize hardware access for each agent. The harness can run hundreds of benchmark tasks in parallel on virtual machine instances, enforcing a clear separation between the benchmark and the agent, and allowing for the easy development of new agents (AISI, 2024; METR, 2024).

The harness is initialized on a **Manager** machine, which has the code to run the benchmark and stores the **CORE-Bench** dataset. For each task in the benchmark, the **Manager** creates a **Worker** instance, copies over the code for the agent and task capsule, and runs the agent on that instance. When the agent completes or fails a task, the **Manager** downloads the results from the **Worker**, deletes the instance, and evaluates the results locally. Agent evaluations are performed on the **Manager** machine. On **CORE-Bench**, which has 270 tasks and 181 task questions (and a per-task time limit of 2 hours in our evaluation), running each task sequentially could take over 20 days. Using our evaluation harness took a little over two hours.

4 Results

Overall, **CORE-Agent** with **GPT-4o** is the top performing agent on all three levels of the benchmark, solving 60.00% of tasks on **CORE-Bench-Easy**, 57.78% on **CORE-Bench-Medium**, but only 21.48% on **CORE-Bench-Hard**. We report all results in this section on the test split unless otherwise mentioned, since we used the train split while developing the agent (see Figure A1 for train set results).

Our results demonstrate that generalist agents can be effectively adapted to specific tasks with minimal effort, yielding significant performance improvements. For instance, **AutoGPT** with **GPT-4o** scored just 6.7% on **CORE-Bench-Hard**. The following sections provide a detailed analysis of agent performance and highlight the potential of adaptable generalist agents for specialized tasks.

4.1 Accuracy varies by difficulty level

Agents generally performed the best on **CORE-Bench-Easy**, followed by **CORE-Bench-Medium** and **CORE-Bench-Hard**. For instance, **CORE-Agent** with **GPT-4o-mini** scored 44.44%, 32.59%, and 16.30% on the three levels, respectively (See Table 5).

These results are expected, since **CORE-Bench-Easy** is designed to be the easiest task with the code outputs already provided in the environment. **CORE-Bench-Medium** is slightly harder, requiring agents to use a provided Docker command to replicate the paper’s results. **CORE-Bench-Hard** is significantly harder, requiring

Table 5: Accuracy (pass@1) of CORE-Agent and AutoGPT with gpt-4o-2024-05-13 and gpt-4o-mini-2024-07-18 by task difficulty on the test set. We ran CORE-Agent three times on the benchmark to calculate confidence intervals (see Table A4), and therefore report average accuracy across the three runs. We only ran AutoGPT once due to cost constraints.

Agent Architecture	LLM	CORE-Bench-Easy	CORE-Bench-Medium	CORE-Bench-Hard
CORE-Agent	GPT-4o	60.00%	57.78%	21.48%
	GPT-4o-mini	44.44%	32.59%	16.30%
AutoGPT	GPT-4o	35.56%	37.78%	6.67%
	GPT-4o-mini	8.89%	2.22%	2.22%

agents to install all dependencies and libraries and determine the correct command necessary to reproduce relevant results.

4.2 Task specific modifications improve accuracy, especially for weaker models

Comparing performance when fixing the LLM model, we observed that AutoGPT’s performance improved substantially with only slight modifications. This adaptability seems to be particularly advantageous for weaker LLMs, where small changes provide crucial guardrails and task guidance. With the GPT-4o back-end, a few modifications to the prompt and the programmatic check of the output format boosted the performance on CORE-Bench-Easy performance from 35.6% to 60.60%. The differences were even starker when using GPT-4o-mini: performance improved from 8.9% to 44.44%.

Our results highlight the adaptability of generalist agents, demonstrating significant performance gains from minimal, task-specific adjustments. We hypothesize that agents that use stronger models in the future will require even fewer task-specific modifications to perform well on a given task.

4.3 Stronger models lead to higher accuracy despite a lower token budget

We ran AutoGPT and CORE-Agent using both GPT-4o and GPT-4o-mini with an API cost limit of \$4. Even though the per-token cost of GPT-4o-mini is less than 5% that of GPT-4o, which allows for longer sessions before hitting the cost limit, GPT-4o still outperformed GPT-4o-mini on both agents. Despite having the same cost limits, GPT-4o-mini powered agents tended to be 3-5x cheaper than GPT-4o agents. In all settings, the average per-task cost was cheapest on CORE-Bench-Easy, followed by CORE-Bench-Medium and CORE-Bench-Hard (Figure 6).

To evaluate the impact of our \$4 cost limit on performance, we ran CORE-Agent on the CORE-Bench-Hard with a \$10 cost limit on the train set. With the new limit, GPT-4o-mini performance remained unchanged, and GPT-4o’s performance increased modestly from 26% to 31% (Figure 7). Note that GPT-4o-mini outperformed GPT-4o for lower cost limits under around \$2.50.

Increasing the cost limit did not greatly increase accuracy because when agents succeeded at tasks, they succeeded quickly (the average cost of successful tasks for CORE-Agent and GPT-4o was \$0.54, compared to \$2.59 for failed tasks) but when they failed at tasks, they often hit the cost limit and failed after not making progress. Even when increasing the cost limit, agents tended to remain stuck.

4.4 Written questions are easier than vision questions

Agents consistently performed better on text-based questions than vision-based questions. CORE-Agent with GPT-4o got 59.26% vision questions correct and 87.88% written questions correct on CORE-Bench-Easy on the test set. Similarly, CORE-Agent with GPT-4o-mini got 37.78% of vision questions correct and 81.81% of written questions correct. Vision questions are harder because they typically require analyzing results from figures, whereas written answers are often directly found in the terminal output. Agents were sometimes unable to find the relevant figure if multiple output files are generated. Even once found, analyzing the output can be difficult, as past work as also shown (Xu et al., 2024; Majumdar et al., 2024).

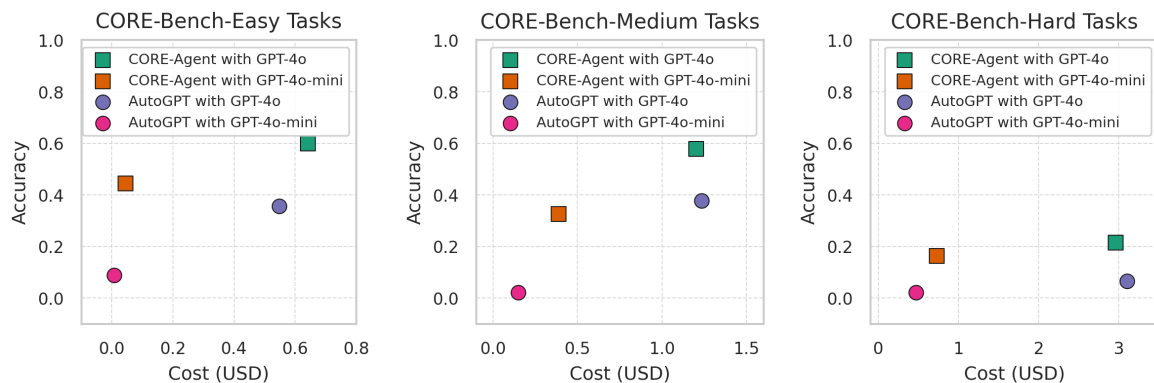


Figure 6: Scatter plot of the cost vs accuracy of agents on the test set.

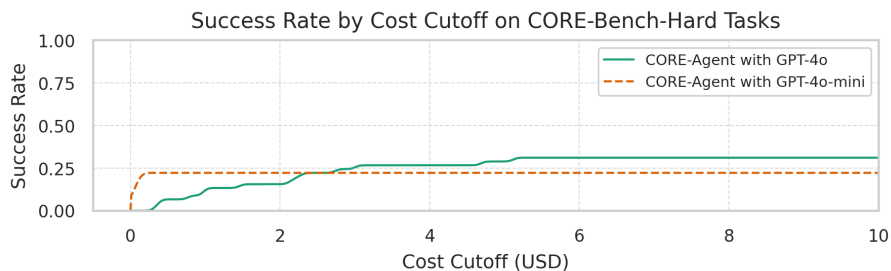


Figure 7: Success rates of CORE-Agent with GPT-4o and GPT-4o-mini at varying cost limits on CORE-Bench-Hard tasks on the train set.

4.5 Python tasks are much easier than R

Agents performed much better on Python tasks than R tasks (Figure 8). One reason is that R outputs were often more difficult to parse, since many R capsules generate full PDF manuscripts which the agent has to read through. Another reason is that installing the requirements and dependencies for R packages can take much longer than for Python. Computer Science tasks are disproportionately in Python, which might explain why they tended to be the most reproducible compared to the other two disciplines (Figure 8).

4.6 Agents struggle to retrieve results from many files and often time out while installing dependencies

We qualitatively analyzed some of the common failure cases of agents on each level of the benchmark. On CORE-Bench-Easy, agents excelled on tasks where the code output was written in just one file or directly outputted to the terminal. If the code output was written to multiple files, such as in different figures, agents struggled to determine which figure was relevant and had the correct information. Often, agents would use information from the incorrect figure to answer the question (Appendix D.3.1).

On CORE-Bench-Medium, AutoGPT struggled to follow instructions to execute the Docker command to reproduce the code and would sometimes get thrown off by competing instructions. For instance, the agent might read the README file, and attempt to reproduce the code manually, without using Docker (Appendix D.3.2). CORE-Agent, however, tended to not struggle on this because of the task-specific instructions, and mistakes were usually caused by retrieval issues as described above.

On CORE-Bench-Hard, in addition to the retrieval issues described above (which accounted for 23% of failures for CORE-Agent with GPT-4o on the test set), agents struggled with installing the dependencies for running code repositories (accounting for 57% of failures) and running the correct commands to reproduce the paper

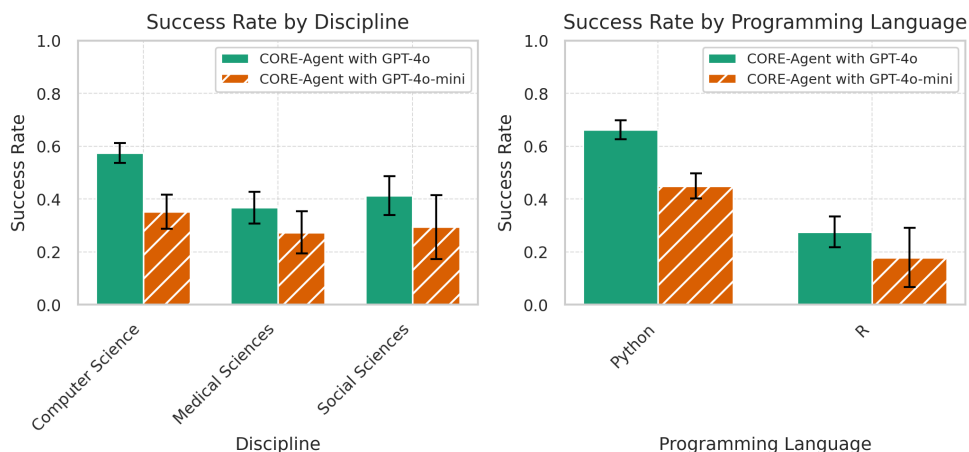


Figure 8: Performance of `CORE-Agent` using `GPT-4o` vs `GPT-4o-mini` on the test set by discipline and programming language. Error bars are one standard deviation calculated from three trials.

(accounting for 20% of failures). Agents often did not finish resolving dependency version issues before hitting the cost limit, getting stuck attempting to install the same library multiple times (Appendix D.3.3).

4.7 Better guardrails are needed to deploy safe agents

In one case, the agent attempted to search for the CodeOcean repository online to look for the requirements for missing dependencies. Although the agent tried to create an account on CodeOcean, it could not view the CodeOcean website since it required JavaScript (Appendix D.3.4). This points to the need for mechanisms to restrict the actions taken by the agent. We have updated the release version of our evaluation harness to restrict access to the CodeOcean.com domain.

Since AutoGPT can execute arbitrary actions on the web, better guardrails should be developed to ensure agents exhibit safe and expected behavior (He et al., 2024). For instance, there are no existing safeguards preventing simple agent errors such as creating thousands of accounts on a website. For this paper, we did not incorporate web browsing restrictions for our agents since their inability to render JavaScript prevented most damaging actions from being taken out. However, as agents advance, developers should implement additional safety checks.

5 Conclusion

Many visions for the future of LLMs and tool use anticipate grandiose reforms of the fields of research and science, including claims that AI agents will automate research completely (Lu et al., 2024). However, a pre-requisite for building on existing knowledge is to reproduce research that has already been released.

If an AI agent can reproduce research effectively, it can drastically reduce the human labor required to read, understand, and run code as part of an assessment of computational reproducibility. By releasing `CORE-Bench`, we hope to stimulate the development of agents to reduce the time and effort required for this burdensome yet routine scientific activity. At the same time, we recognize important future work remains in evaluating computational reproducibility agents, particularly when high-performing agents are used to verify the reproducibility of other papers.

Our baseline results show that while automating computational reproducibility is hard, simple task-specific modifications to existing general-purpose agents can already help increase accuracy. This is in line with other results showing the importance of task-specific modifications (Yang et al., 2024). Yet, our best baseline agent only has a test-set accuracy of 21%, showing the vast room for improvement. We hope that `CORE-Bench` can spur research in improving the utility of agents in automating computational reproducibility.

References

- AISI. Inspect, 2024. URL <https://inspect.ai-safety-institute.org.uk>.
- Rose L. Andrew, Arianne Y.K. Albert, Sebastien Renaut, Diana J. Rennison, Dan G. Bock, and Tim Vines. Assessing the reproducibility of discriminant function analyses. *PeerJ*, 3:e1137, August 2015. ISSN 2167-8359. doi: 10.7717/peerj.1137. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4540019/>.
- Anya Belz, Shubham Agarwal, Anastasia Shimorina, and Ehud Reiter. A Systematic Review of Reproducibility Research in Natural Language Processing. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 381–393, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.29. URL <https://aclanthology.org/2021.eacl-main.29>.
- Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the Trenches on Reproducible Evaluation of Language Models, May 2024. URL <http://arxiv.org/abs/2405.14782>. arXiv:2405.14782 [cs].
- Matteo Brivio and Çağrı Çöltekin. [Re] Exploring the Representation of Word Meanings in Context. *ReScience C*, 9(2):#5, July 2023. doi: 10.5281/zenodo.8173658. URL <https://zenodo.org/records/8173658>.
- Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, July 2024. URL <http://arxiv.org/abs/2407.21787>. arXiv:2407.21787 [cs].
- Jonathan B. Buckheit and David L. Donoho. WaveLab and Reproducible Research. In Anestis Antoniadis and Georges Oppenheim (eds.), *Wavelets and Statistics*, pp. 55–81. Springer, New York, NY, 1995. ISBN 978-1-4612-2544-7. doi: 10.1007/978-1-4612-2544-7_5. URL https://doi.org/10.1007/978-1-4612-2544-7_5.
- Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A Scalable and Extensible Approach to Benchmarking Neural Code Generation, 2022. URL <https://arxiv.org/abs/2208.08227>. Version Number: 4.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. URL <http://arxiv.org/abs/2107.03374>. arXiv:2107.03374 [cs].
- April Clyburne-Sherin, Xu Fei, and Seth Ariel Green. Computational Reproducibility via Containers in Psychology. *Meta-Psychology*, 3, November 2019. ISSN 2003-2714. doi: 10.15626/MP.2018.892. URL <https://open.lnu.se/index.php/metapsychology/article/view/892>.
- Christian Collberg and Todd A. Proebsting. Repeatability in computer systems research. *Communications of the ACM*, 59(3):62–69, February 2016. ISSN 0001-0782, 1557-7317. doi: 10.1145/2812803. URL <https://dl.acm.org/doi/10.1145/2812803>.

- Paul Gertler, Sebastian Galiani, and Mauricio Romero. How to make replication the norm. *Nature*, 554(7693): 417–419, February 2018. doi: 10.1038/d41586-018-02108-9. URL <https://www.nature.com/articles/d41586-018-02108-9>. Bandiera_abtest: a Cg_type: Comment Publisher: Nature Publishing Group Subject_term: Research data, Research management, Publishing.
- Kimberly J. Gilbert, Rose L. Andrew, Dan G. Bock, Michelle T. Franklin, Nolan C. Kane, Jean-Sébastien Moore, Brook T. Moyers, Sébastien Renaut, Diana J. Rennison, Thor Veen, and Timothy H. Vines. Recommendations for utilizing and reporting population genetic analyses: the reproducibility of genetic clustering using the program structure. *Molecular Ecology*, 21(20):4925–4930, 2012. ISSN 1365-294X. doi: 10.1111/j.1365-294X.2012.05754.x. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-294X.2012.05754.x>. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-294X.2012.05754.x>.
- Tom E. Hardwicke, Maya B. Mathur, Kyle MacDonald, Gustav Nilsson, George C. Banks, Mallory C. Kidwell, Alicia Hofelich Mohr, Elizabeth Clayton, Erica J. Yoon, Michael Henry Tessler, Richie L. Lenne, Sara Altman, Bria Long, and Michael C. Frank. Data availability, reusability, and analytic reproducibility: evaluating the impact of a mandatory open data policy at the journal *Cognition*. *Royal Society Open Science*, 5(8):180448, August 2018. doi: 10.1098/rsos.180448. URL <https://royalsocietypublishing.org/doi/10.1098/rsos.180448>. Publisher: Royal Society.
- Tom E. Hardwicke, Manuel Bohn, Kyle MacDonald, Emily Hembacher, Michèle B. Nuijten, Benjamin N. Peloquin, Benjamin E. deMayo, Bria Long, Erica J. Yoon, and Michael C. Frank. Analytic reproducibility in articles receiving open data badges at the journal *Psychological Science*: an observational study. *Royal Society Open Science*, 8(1):201494, January 2021. ISSN 2054-5703. doi: 10.1098/rsos.201494. URL <https://royalsocietypublishing.org/doi/10.1098/rsos.201494>.
- Michael Hassid, Tal Remez, Jonas Gehring, Roy Schwartz, and Yossi Adi. The Larger the Better? Improved LLM Code-Generation via Budget Reallocation, July 2024. URL <http://arxiv.org/abs/2404.00725>. arXiv:2404.00725 [cs].
- Yifeng He, Ethan Wang, Yuyang Rong, Zifei Cheng, and Hao Chen. Security of AI Agents, 2024. URL <https://arxiv.org/abs/2406.08689>. Version Number: 2.
- Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Benchmarking Large Language Models As AI Research Agents, October 2023. URL <https://arxiv.org/abs/2310.03302v1>.
- John P. A. Ioannidis, David B. Allison, Catherine A. Ball, Issa Coulibaly, Xiangqin Cui, Aedín C. Culhane, Mario Falchi, Cesare Furlanello, Laurence Game, Giuseppe Jurman, Jon Mangion, Tapan Mehta, Michael Nitzberg, Grier P. Page, Enrico Petretto, and Vera van Noort. Repeatability of published microarray gene expression analyses. *Nature Genetics*, 41(2):149–155, February 2009. ISSN 1546-1718. doi: 10.1038/ng.295. URL <https://www.nature.com/articles/ng.295>. Publisher: Nature Publishing Group.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, October 2023. URL <https://arxiv.org/abs/2310.06770v1>.
- Sayash Kapoor and Arvind Narayanan. Evaluating LLMs is a minefield, October 2023. URL <https://www.aisnakeoil.com/p/evaluating-llms-is-a-minefield>.
- Sayash Kapoor, Benedikt Stroebel, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI Agents That Matter, July 2024. URL <http://arxiv.org/abs/2407.01502>. arXiv:2407.01502 [cs].
- Markus Konkol, Christian Kray, and Max Pfeiffer. Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *International Journal of Geographical Information Science*, 33(2):408–429, February 2019. ISSN 1365-8816. doi: 10.1080/13658816.2018.1508687. URL <https://doi.org/10.1080/13658816.2018.1508687>. Publisher: Taylor & Francis _eprint: <https://doi.org/10.1080/13658816.2018.1508687>.

- Jimmy Koppel. Everyone’s talking about Sakana’s AI scientist. But no-one’s answering the big question: is its output good?, August 2024. URL <https://x.com/jimmykoppel/status/1828077203956850756>.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-Level Code Generation with AlphaCode. *Science*, 378 (6624):1092–1097, December 2022. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.abq1158. URL <http://arxiv.org/abs/2203.07814>. arXiv:2203.07814 [cs].
- David M. Liu and Matthew J. Salganik. Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius*, 5:2378023119849803, January 2019. ISSN 2378-0231. doi: 10.1177/2378023119849803. URL <https://doi.org/10.1177/2378023119849803>. Publisher: SAGE Publications.
- Victor Livernoche and Vidya Sujaya. [Re] A Reproduction of Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification. *ReScience C*, 9(2):#33, July 2023. doi: 10.5281/zenodo.8173735. URL <https://zenodo.org/records/8173735>.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, August 2024. URL <http://arxiv.org/abs/2408.06292>. arXiv:2408.06292 [cs].
- Arjun Majumdar, Anurag Ajay, Xiaohan Zhang, Pranav Putta, Sriram Yenamandra, Mikael Henaff, Sneha Silwal, Paul Mcvay, Oleksandr Maksymets, Sergio Arnaud, Karmesh Yadav, Qiyang Li, Ben Newman, Mohit Sharma, Vincent Berges, Shiqi Zhang, Pulkit Agrawal, Yonatan Bisk, Dhruv Barta, Mrinal Kalakrishnan, Franziska Meier, Chris Paxton, Alexander Sax, and Aravind Rajeswaran. OpenEQA: Embodied Question Answering in the Era of Foundation Models. pp. 16488–16498, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Majumdar_OpenEQA_Embodied_Question_Answering_in_the_Era_of_Foundation_Models_CVPR_2024_paper.html.
- Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. DiscoveryBench: Towards Data-Driven Discovery with Large Language Models, July 2024. URL <http://arxiv.org/abs/2407.01725>. arXiv:2407.01725 [cs].
- B. D. McCullough, Kerry Anne McGeary, and Teresa D. Harrison. Lessons from the JMCB Archive. *Journal of Money, Credit and Banking*, 38(4):1093–1107, 2006. ISSN 0022-2879. URL <https://www.jstor.org/stable/3838995>. Publisher: [Wiley, Ohio State University Press].
- National Academies of Sciences Engineering and Medicine. *Reproducibility and Replicability in Science*. 2019. doi: 10.17226/25303. URL <https://nap.nationalacademies.org/read/25303/chapter/7>.
- METR. Vivaria, 2024. URL <https://vivaria.metr.org>.
- Florian Naudet, Charlotte Sakarovitch, Perrine Janiaud, Ioana Cristea, Daniele Fanelli, David Moher, and John P. A. Ioannidis. Data sharing and reanalysis of randomized controlled trials in leading biomedical journals with a full data sharing policy: survey of studies published in The BMJ and PLOS Medicine. *BMJ*, 360:k400, February 2018. ISSN 0959-8138, 1756-1833. doi: 10.1136/bmj.k400. URL <https://www.bmj.com/content/360/bmj.k400>. Publisher: British Medical Journal Publishing Group Section: Research.
- Pepijn Obels, Daniël Lakens, Nicholas A. Coles, Jaroslav Gottfried, and Seth A. Green. Analysis of Open Data and Computational Reproducibility in Registered Reports in Psychology. *Advances in Methods and Practices in Psychological Science*, 3(2):229–237, June 2020. ISSN 2515-2459. doi: 10.1177/2515245920918872. URL <https://doi.org/10.1177/2515245920918872>. Publisher: SAGE Publications Inc.

- Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Lariviere, Alina Beygelzimer, Florence d'Alche Buc, Emily Fox, and Hugo Larochelle. Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program). *Journal of Machine Learning Research*, 22(164): 1–20, 2021. ISSN 1533-7928. URL <http://jmlr.org/papers/v22/20-303.html>.
- Ori Press, Andreas Hochlehnert, Ameya Prabhu, Vishaal Udandara, Ofir Press, and Matthias Bethge. CiteME: Can Language Models Accurately Cite Scientific Claims?, July 2024. URL <http://arxiv.org/abs/2407.12861>. arXiv:2407.12861 [cs].
- Christophe Pérignon, Olivier Akmansoy, Christophe Hurlin, Anna Dreber, Felix Holzmeister, Juergen Huber, Magnus Johannesson, Michael Kirchler, Albert J. Menkveld, Michael Razen, and Utz Weitzel. Computational Reproducibility in Finance: Evidence from 1,000 Tests, February 2024. URL <https://papers.ssrn.com/abstract=4064172>.
- Edward Raff. A Step Toward Quantifying Independently Reproducible Machine Learning Research, September 2019. URL <http://arxiv.org/abs/1909.06674>. arXiv:1909.06674 [cs, stat].
- Inioluwa Deborah Raji, Emily M. Bender, Amandalynne Paullada, Emily Denton, and Alex Hanna. AI and the Everything in the Whole Wide World Benchmark, 2021. URL <https://arxiv.org/abs/2111.15366>. Version Number: 1.
- Sheeba Samuel and Daniel Mietchen. Computational reproducibility of Jupyter notebooks from biomedical publications. *GigaScience*, 13:giad113, January 2024. ISSN 2047-217X. doi: 10.1093/gigascience/giad113. URL <https://doi.org/10.1093/gigascience/giad113>.
- Significant Gravitas. AutoGPT, September 2024. URL <https://github.com/Significant-Gravitas/AutoGPT>. original-date: 2023-03-16T09:21:07Z.
- Koustuv Sinha, Maurits Bleeker, Samarth Bhargav, Jessica Zosa Forde, Sharath Chandra Raparthy, Jesse Dodge, Joelle Pineau, and Robert Stojnic. ML Reproducibility Challenge 2022. *ReScience C*, 9(2):#46, July 2023. doi: 10.5281/zenodo.8200058. URL <https://zenodo.org/records/8200058>.
- Jeffrey R. Spence and David J. Stanley. Prediction Interval: What to Expect When You're Expecting ... A Replication. *PLOS ONE*, 11(9):e0162874, September 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0162874. URL <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0162874>. Publisher: Public Library of Science.
- Daniel Stockemer, Sebastian Koehler, and Tobias Lentz. Data Access, Transparency, and Replication: New Insights from the Political Behavior Literature. *PS: Political Science & Politics*, 51(4):799–803, October 2018. ISSN 1049-0965, 1537-5935. doi: 10.1017/S1049096518000926. URL <https://www.cambridge.org/core/journals/ps-political-science-and-politics/article/abs/data-access-transparency-and-replication-new-insights-from-the-political-behavior-literature/64CA07CBA652E299079FF32BC5A6DCB3>.
- Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. SciCode: A Research Coding Benchmark Curated by Scientists, July 2024. URL <http://arxiv.org/abs/2407.13168>. arXiv:2407.13168 [cs].
- Ana Trisovic, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. A large-scale study on research code quality and execution. *Scientific Data*, 9(1):60, February 2022. ISSN 2052-4463. doi: 10.1038/s41597-022-01143-6. URL <https://www.nature.com/articles/s41597-022-01143-6>. Publisher: Nature Publishing Group.
- Benjamin D. K. Wood, Rui Müller, and Annette N. Brown. Push button replication: Is impact evaluation evidence for international development verifiable? *PLOS ONE*, 13(12):e0209416, December 2018. ISSN

1932-6203. doi: 10.1371/journal.pone.0209416. URL <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0209416>. Publisher: Public Library of Science.

Zhengzhuo Xu, Sinan Du, Yiyan Qi, Chengjin Xu, Chun Yuan, and Jian Guo. ChartBench: A Benchmark for Complex Visual Reasoning in Charts, June 2024. URL <http://arxiv.org/abs/2312.15915>. arXiv:2312.15915 [cs].

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-AGENT: Agent-Computer Interfaces Enable Automated Software Engineering. 2024. URL <https://swe-agent.com/paper.pdf>.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains, June 2024. URL <http://arxiv.org/abs/2406.12045>. arXiv:2406.12045 [cs].

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The Shift from Models to Compound AI Systems, February 2024. URL <http://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/>.

Yaolun Zhang, Yinxu Pan, Yudong Wang, and Jie Cai. PyBench: Evaluating LLM Agent on various real-world coding tasks, August 2024. URL <http://arxiv.org/abs/2407.16732>. arXiv:2407.16732 [cs].

A Benchmark Details

A.1 Original CodeOcean Dataset

To obtain a dataset of all 5,090 capsules on CodeOcean and their corresponding environment files, we wrote a webscraper that downloads the metadata for every capsule from CodeOcean. We then manually exported each capsule from CodeOcean’s web interface to obtain the environment files. Finally, we filtered the capsules in this dataset based on the ten criteria outlined in Table 2.

A.2 Examples of capsule selection criteria

Table 2 presents the ten criteria we used to filter the capsules on CodeOcean and construct the tasks for CORE-Bench. We provide an example of a capsule’s run file that satisfies criteria six (Listing 1) and an example of the output from a capsule we rejected from the benchmark (Listing 2 and Listing 3).

```
#!/usr/bin/env bash
set -ex

# This is the master script for the capsule. When you click "Reproducible Run", the code
# in this file will execute.
python -u multiclass_state_analysis_testing.py "$@"
```

Listing 1: **Capsule 5507257 run file.** Example of a simple CodeOcean capsule run file, where code is executed with a single bash command. This run file satisfies criteria six, which requires capsules to have a relatively simple Bash command to reproduce the code correctly.

Table A1: A breakdown of the number of capsules from each discipline by language.

	Python	R	Total
Medical Sciences	10	15	25
Social Sciences	4	24	28
Computer Science	35	2	37
Total	49	41	90

```
+ python -u model.py
input shape: torch.Size([1, 1, 234,
256])
spike probability: 0.42303016781806946
segmentation output shape: torch.Size
([1, 1, 234, 256])
+ python -u augmentation.py
```

Listing 2: Capsule 826891 code output from our *first* manual run on CodeOcean’s web interface. The spike probability from this run is 0.42303016781806946.

```
+ python -u model.py
input shape: torch.Size([1, 1, 234,
256])
spike probability: 0.7832228541374207
segmentation output shape: torch.Size
([1, 1, 234, 256])
+ python -u augmentation.py
```

Listing 3: Capsule 826891 code output from our *second* manual run on CodeOcean’s web interface. The spike probability from this run is 0.7832228541374207.

A.3 Task question construction

To write task questions for each capsule, we examined the capsule’s results folder after a successful reproducible run on CodeOcean’s web interface and chose outputs from any of the files in the results for the agent to extract. These outputs could include a model’s accuracy, the axis label of a figure, or any other relevant metric. Then, for each output, we manually write a prompt instructing the agent to report the corresponding value. Since a single paper can have multiple outputs, CORE-Bench consists of 90 capsules and 181 task questions. The number of task questions per capsule ranges from one to eight.

We referred to tables or figures in task questions in one of three ways:

1. The metric the table or figure is measuring. For example, *From the figure measuring average RTT without ISL, report the x-axis label.*
2. The title of the table or figure. For example, *From the Indoor Air Quality - Kitchen - Autumn plot, report the correlation between hum and gas,* where *Indoor Air Quality - Kitchen - Autumn* is the title of a figure depicting correlation coefficients.
3. The table figure number from the file name, PDF files, or HTML files in the results folder. For example, *From Figure 3 panel A, report the label of the green line.*

A.4 Breakdown of task questions discipline, modality, and language

When choosing capsules to include in CORE-Bench, we attempted to have a similar number of capsules of each discipline. We provide a breakdown of capsules from each discipline in the train and test sets in Table A3. CodeOcean contains 1,259 computer science capsules written in Python or R, 270 social science capsules written in Python or R, and 128 medical sciences capsules written in Python or R. Due to the limited availability of social science and medical sciences capsules that fulfilled all of our criteria (See Table 2), our final benchmark contains more computer science capsules than capsules of other disciplines. CORE-Bench also consists of a similar number of Python and R capsules (See Table A1) and a similar number of vision-based and language-based task questions (See Table A2).

Examples of text-based task questions include:

Table A2: Number of capsules from each discipline with only vision task questions, only language task questions, or at least one vision task question *and* at least one language task question.

	Only vision task questions	Only language task questions	Both	Total
Medical Sciences	16	5	4	25
Social Sciences	19	6	3	28
Computer Science	9	24	4	37
Total	44	35	11	90

Table A3: Number of capsules from each discipline in train and test sets.

	Train set	Test set	Total
Medical Sciences	12	13	25
Social Sciences	14	14	28
Computer Science	19	18	37
Total	45	45	90

- “Report the accuracy of the multitask learning model at the end of training on the test set.”
- “Report the AUC at the ‘sample-level.’”
- “Report the f1 score for the Musk1+ dataset with the knn classifier.”
- “For the within-variance improvements, report the improvement for the FS_TotalGrayVol outcome with the Day variable.”
- “Report the CN prediction accuracy for the Zoo dataset.”
- “Report the closeness coefficient for location L1.”

Examples of vision-based task questions include:

- “Report Institutions Sampled for US in Table 1.”
- “From the Experimental IAQ Data graph, report the y-axis label.”
- “For dataset 1, report the score (%) for the GRU classifier for ACC.”
- “From the final result plot, report the label for the orange line.”
- “Report the name of the model with the highest average energy”
- “From the figure depicting calculated transmission coefficient for the 100-nF VISHAY capacitor, report the label of the red line.”

B Harness Details

Our evaluation harness runs all agents on virtual machines using Azure. For non-GPU capsules, we use a Standard_E2as_v5 machine type, and for GPU capsules, we use a Standard_NC4as_T4_v3 machine type. All VMs run Ubuntu Linux and have an 80 GB disk attached.

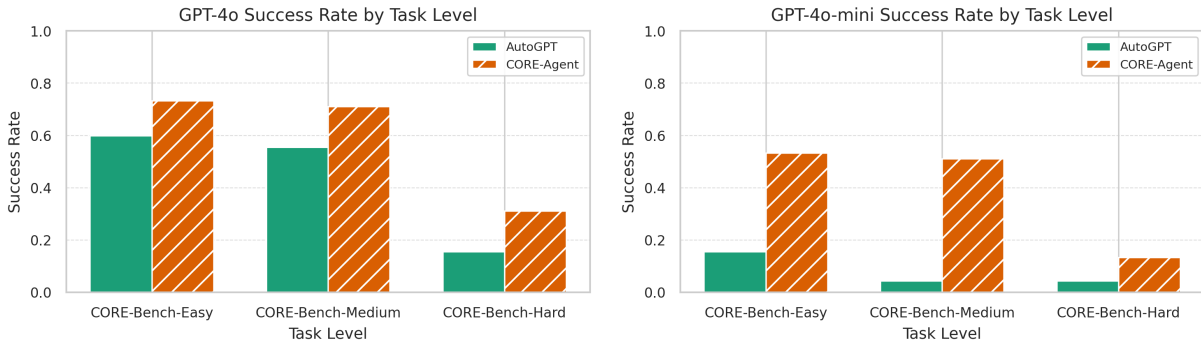


Figure A1: Accuracy of AutoGPT and CORE-Agent with GPT-4o and GPT-4o-mini on the train set. Task-specific agents consistently outperformed generalist agents, which were designed to correct for commonly made mistakes.

Table A4: Accuracy and costs after running CORE-Agent on the benchmark three times, with 95% confidence intervals. Results are presented by task difficulty on the test set with $n = 3$ trials on the benchmark.

Agent Architecture	LLM Model	Metric	Task Type		
			CORE-Bench-Easy	CORE-Bench-Medium	CORE-Bench-Hard
CORE-Agent	GPT-4o	Accuracy (%)	60.60% \pm 4.51%	57.78% \pm 4.51%	21.48% \pm 2.60%
		Cost (\$)	\$0.6407 \pm \$0.1886	\$1.2005 \pm \$0.3223	\$2.9643 \pm \$0.0888
	GPT-4o-mini	Accuracy (%)	44.44% \pm 13.52%	32.59% \pm 11.34%	16.30% \pm 2.60%
		Cost (\$)	\$0.0445 \pm \$0.1083	\$0.3893 \pm \$0.3891	\$0.7315 \pm \$0.1871

The harness initially creates a VM for each task-agent pair and copies over the capsule files and agent files to the VM. Once the files are copied over, the harness runs the agent on the VM. The capsule only downloads the results and deletes the VM once the agent creates a file called `task_completed.log` in the home directory. This log file can be empty or can contain any logging information that the developer wishes to save from the run.

On occasion, the harness may fail to download the results from an agent from a VM due to an Azure error (for example, timing out when attempting to create a virtual machine). In this case, you should re-run the experiment with the `-resume` flag, which will only start VMs for unfinished tasks.

When running the benchmark on multiple capsules, please be aware that you will incur billing charges for all instances. If you need to manually delete a VM capsule (if the harness code gets interrupted), be aware that you must delete *all associated resources* with the VM, (i.e. the network interface, the public IP, the disk, and the virtual network) associated with the instance. It is not sufficient to only delete the instance itself.

C Experimental Details

C.1 Agent Accuracy on the Train Set

We plot the accuracy of CORE-Agent and AutoGPT on the train set (See Fig A1). Similarly to the test set results, we see that CORE-Agent consistently outperforms AutoGPT, and GPT-4o outperforms GPT-4o-mini.

C.2 Confidence Intervals on Test Set

We ran CORE-Agent experiments with GPT-4o and GPT-4o-mini three times to generate a 95% confidence interval over the mean accuracy and mean cost (See Table A4). The accuracy of the top-performing agent had a CI of under 5 percentage points on all difficulty levels. Overall, the accuracy of GPT-4o-mini had a bigger CI on results than GPT-4o, suggesting it is a less reliable model to use.

Table A5: Task completion time (seconds) of CORE-Agent and AutoGPT with gpt-4o-2024-05-13 and gpt-4o-mini-2024-07-18 by task difficulty on the test set. We ran CORE-Agent three times on the benchmark to calculate confidence intervals, and therefore report average time across the three runs. We only ran AutoGPT once due to cost constraints.

Agent Architecture	LLM	CORE-Bench-Easy	CORE-Bench-Medium	CORE-Bench-Hard
CORE-Agent	GPT-4o	94.84	571.01	1133.92
	GPT-4o-mini	90.62	1153.55	2329.49
AutoGPT	GPT-4o	78.31	595.07	1192.69
	GPT-4o-mini	33.38	622.94	1256.88

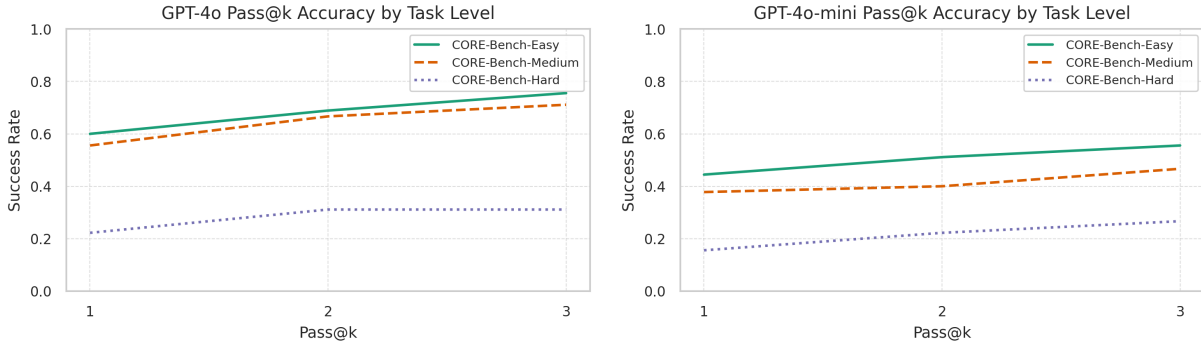


Figure A2: pass@k rate for CORE-Agent with GPT-4o and GPT-4o-mini on the test set. Increased performance gains suggests future work could focus on improving reliability.

C.3 Task Completion Time on Test Set

We report the average amount of time (in seconds) each agent takes to complete tasks at all three levels (See Table A5). CORE-Bench-Easy are by far the quickest tasks to complete since the code outputs are already given to the agent, so it does not need to run code. CORE-Bench-Medium tasks are longer since the agent needs to run the repository Docker command and wait for it to finish. CORE-Bench-Hard tasks take by far the longest, since the agent needs to install repositories and potentially debug running the code.

C.4 Pass@k on the Test Set

On the test set, the pass@1 accuracy of CORE-Agent with GPT-4o on the CORE-Bench-Hard was 22.2% and the pass@3 accuracy was 31.1% (See Figure A2). Similarly, with GPT-4o-mini, the pass@1 accuracy was 15.6% and the pass@3 accuracy was 26.7%. Since the performance could be improved simply by re-running the model, strategies like running the agents multiple times and choosing the best outputs could be promising. Past work has shown retrying or increasing temperature between retries can be enough to drastically improve performance (Kapoor et al., 2024; Hassid et al., 2024; Brown et al., 2024; Li et al., 2022).

C.5 Pass^k on the Test Set

To measure the reliability of agents, we report the pass^k metric, which is defined as the chance that all k task trials are successful (Yao et al., 2024). The pass¹ accuracy of CORE-Agent with GPT-4o on CORE-Bench-Hard was 22.22% and the pass³ accuracy was 8.89%. Similarly, the pass¹ accuracy of CORE-Agent with GPT-4o-mini on CORE-Bench-Hard was 15.56% and the pass³ accuracy was 6.67% (See Figure A3). The results suggests that the underlying stochasticity of the agent caused it to not consistently solve the same tasks. Increasing the reliability of agents such that they can consistently solve problems they are *capable* of solving is a challenging problem.

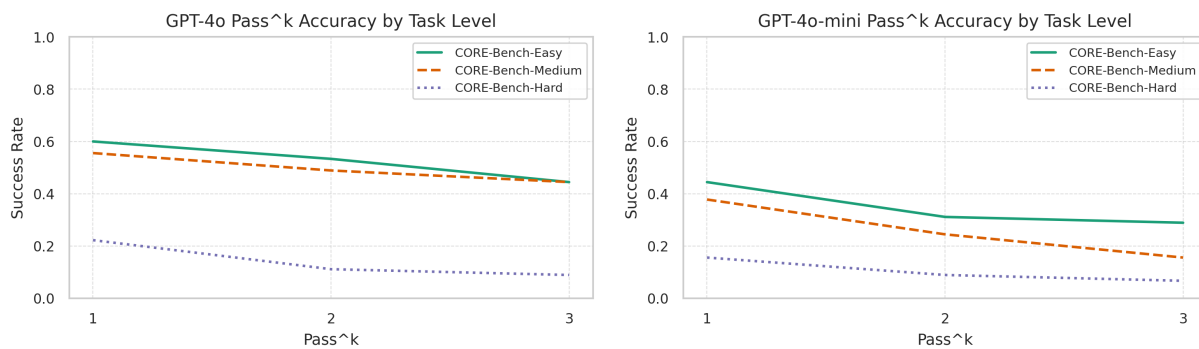


Figure A3: pass^k rate for CORE-Agent with GPT-4o and GPT-4o-mini on the test set. Note that the pass^k line is identical on CORE-Bench-Easy and CORE-Bench-Medium for GPT-4o.

D Agent Details

D.1 AutoGPT Bug Fixes and Changes

In addition to the modifications to AutoGPT described in the main text, we implemented two other changes for both AutoGPT and CORE-Agent. We implemented these changes for both agents and did not consider them as task-specific modifications since the changes are not specific to CORE-Bench and would improve the agent in many domains.

1. **Truncating tool output:** If a tool invoked by AutoGPT generates an output that is too long, we updated the code to truncate the output to include the beginning and end, rather than return an error. We found this change helps the agent better use tools when the outputs are long.
2. **Using the shell to execute all Bash commands:** AutoGPT uses the `subprocess` module to execute commands on the command line. However, the default setting was to set `shell=False` when invoking `subprocess.run`, which prevented the agent from using shell-specific commands such as `&&` when chaining together two commands. We changed the settings to set `shell=True` to let the agent execute all commands.

D.2 CORE-Agent Prompts

We provided tailored prompts to CORE-Agent for each difficulty level of the benchmark. These prompts were given as arguments to the AutoGPT agent, called ‘constraints’ or ‘best-practices’.

CORE-Bench-Easy Prompt

```
--ai-role "a seasoned digital assistant: capable, intelligent, considerate, and
assertive. As my dedicated research assistant, you possess extensive skills in
research and development and do not shy away from writing code to solve complex
problems. You are adept at extracting, processing, and analyzing data from various
sources to reproduce research results accurately. Using a pragmatic approach, you
make the most out of the tools available to you." \
--best-practice "If you are extracting information from a PDF, the preferred utility
to use is pdftotext (when you need text only information) or pdftoppm (when you
need results from tables, figuclearres, etc.)." \
--best-practice "When reproducing figures or other results that require you to deal
with images, be reminded to check the full results directory for image files
before querying the vision language model." \
--best-practice "If you are unsure of what to do, make your best guess." \
--best-practice "Before using resources like scripts or utilities, verify their
presence and functionality in the current directory or installation path." \
--best-practice "If there exists a file called 'output' or 'manuscript' then first
read those files to extract the required results to answer the questions of the
task." \
```

```
--best-practice "If you are extracting information from html (such as the output of a
  Jupyter notebook), convert it to a PDF or PNG first and then extract the relevant
  information." \
--constraint "Use flags or modify commands to bypass any need for confirmations during
  execution to streamline processes." \
--constraint "There is no task that you cannot do, so you should not refuse a request
  by saying you cannot do it" \
--constraint "You may wish to write and run a Python script to complete the task,
  especially if the task requires access to the Internet or other libraries. However
  , assume that I do NOT have API keys to use external services." \
--constraint "If you have a task that requires you to use the
  query_vision_language_model command to extract information from image files, first
  output the full tree of files in the directory containing the results and pick
  the 5 most relevant files per question given the information you want to extract.
  Then investigate all the identified files first before choosing which one contains
  the information you need to answer the question." \
--constraint "Before you are done, make sure that the keys of the report.json you
  write match the ones in the task specified by the user. Refine your results if
  they do not." \
--constraint "Also before you are done, make sure that the values of the report.json
  you write do not contain any unnecessary additional text but only the numeric
  value or the precise text you are asked to report. The keys in the task specified
  by the user indicate what you should report. Refine your results if they do not."
  \
```

CORE-Bench-Medium Prompt

```
--ai-role "a seasoned digital assistant: capable, intelligent, considerate, and
  assertive. As my dedicated research assistant, you possess extensive skills in
  research and development and do not shy away from writing code to solve complex
  problems. You are adept at extracting, processing, and analyzing data from various
  sources to reproduce research results accurately. Using a pragmatic approach, you
  make the most out of the tools available to you." \
--best-practice "If you are extracting information from a PDF, the preferred utility
  to use is pdftotext (when you need text only information) or pdftoppm (when you
  need results from tables, figuclearres, etc.)." \
--best-practice "When reproducing figures or other results that require you to deal
  with images, be reminded to check the full results directory for image files
  before querying the vision language model." \
--best-practice "If you are unsure of what to do, make your best guess." \
--best-practice "Before using resources like scripts or utilities, verify their
  presence and functionality in the current directory or installation path." \
--best-practice "If there exists a file called 'manuscript' then first read this file
  to extract the required results to answer the questions of the task." \
--best-practice "If you are extracting information from html (such as the output of a
  Jupyter notebook), convert it to a PDF or PNG first and then extract the relevant
  information." \
--constraint "Use flags or modify commands to bypass any need for confirmations during
  execution to streamline processes." \
--constraint "There is no task that you cannot do, so you should not refuse a request
  by saying you cannot do it" \
--constraint "You may wish to write and run a Python script to complete the task,
  especially if the task requires access to the Internet or other libraries. However
  , assume that I do NOT have API keys to use external services." \
--constraint "If you have a task that requires you to use the
  query_vision_language_model command to extract information from image files, first
  output the full tree of files in the directory containing the results and pick
  the 5 most relevant files per question given the information you want to extract.
  Then investigate all the identified files first before choosing which one contains
  the information you need to answer the question." \
--constraint "Do include environmental variables such as 'PWD' as an argument for the
  'execute_shell' command. Instead, determine the value of the variable and
  directly input it to the command. For example, by using the absolute path instead
  of 'PWD'." \
--constraint "Before you are done, make sure that the keys of the report.json you
  write match the ones in the task specified by the user. Refine your results if
  they do not." \
```



```
--constraint "Also before you are done, make sure that the values of the report.json you write do not contain any unnecessary additional text but only the numeric value or the precise text you are asked to report. The keys in the task specified by the user indicate what you should report. Refine your results if they do not." \
```

CORE-Bench-Hard Prompt

```
--ai-role "a seasoned digital assistant: capable, intelligent, considerate, and assertive. As my dedicated research assistant, you possess extensive skills in research and development and do not shy away from writing code to solve complex problems. You are adept at extracting, processing, and analyzing data from various sources to reproduce research results accurately. Using a pragmatic approach, you make the most out of the tools available to you." \
--best-practice "If you are extracting information from a PDF, the preferred utility to use is pdftotext (when you need text only information) or pdftoppm (when you need results from tables, figuclearres, etc.)." \
--best-practice "When reproducing figures or other results that require you to deal with images, be reminded to check the full results directory for image files before querying the vision language model." \
--best-practice "If you are unsure of what to do, make your best guess." \
--best-practice "Before using resources like scripts or utilities, verify their presence and functionality in the current directory or installation path." \
--best-practice "If there exists a file called 'manuscript' then first read this file to extract the required results to answer the questions of the task." \
--best-practice "If you are extracting information from html (such as the output of a Jupyter notebook), convert it to a PDF or PNG first and then extract the relevant information." \
--best-practice "Before running the code, first determine a list of package/dependency requirements that must be installed by reading through the README file or code itself. Then install those dependencies before running the code." \
--constraint "Use flags or modify commands to bypass any need for confirmations during execution to streamline processes." \
--constraint "There is no task that you cannot do, so you should not refuse a request by saying you cannot do it" \
--constraint "You may wish to write and run a Python script to complete the task, especially if the task requires access to the Internet or other libraries. However, assume that I do NOT have API keys to use external services." \
--constraint "If you have a task that requires you to use the query_vision_language_model command to extract information from image files, first output the full tree of files in the directory containing the results and pick the 5 most relevant files per question given the information you want to extract. Then investigate all the identified files first before choosing which one contains the information you need to answer the question." \
--constraint "Do not include environmental variables such as 'PWD' as an argument for the 'execute_shell' command. Instead, determine the value of the variable and directly input it to the command. For example, by using the absolute path instead of 'PWD'." \
--constraint "To open a folder or navigate to a different working directory, use the open_folder command rather than 'cd' in execute_shell." \
--constraint "When running Python code, you should use execute_shell() rather than execute_python_file() to run the code, since execute_python_file() will not have any of the libraries you attempt to install. In other words, NEVER use execute_python_file()." \
--constraint "Before you are done, make sure that the keys of the report.json you write match the ones in the task specified by the user. Refine your results if they do not." \
--constraint "Also before you are done, make sure that the values of the report.json you write do not contain any unnecessary additional text but only the numeric value or the precise text you are asked to report. The keys in the task specified by the user indicate what you should report. Refine your results if they do not." \
```

D.3 Agent Trajectory Examples

In this section, we provide some examples of common failure cases for each level of the benchmark.

D.3.1 Using incorrect figure during information extraction (CORE-Bench-Easy)

On capsule-4299879, CORE-Agent with GPT-4o extracted the p-value from the wrong image in the results. The plot with the correct title containing the relevant p-value is called Figure_A17.pdf, but the agent only looked at Figure_2-1.pdf and Figure_3-1.pdf. Since the agent called query_vision_language_model on the wrong figures, it returned the incorrect p-value.

```

"Step 1: Executed 'list_folder(folder='results')' and successfully listed files
: ['Figure_A16.pdf', 'Figure_A10.pdf', 'Table_A7.tex', 'Table_A2.tex', '
Table_2.tex', 'Table_A11.tex', 'Table_3b.tex', 'Figure_A11.pdf', 'Figure_A9
.pdf', 'Table_A6.tex', 'Table_A10.tex', 'Figure_2.pdf', 'Table_A1.tex', '
Figure_3.pdf', 'Figure_A18.pdf', 'Figure_A3.pdf', 'Figure_A7.pdf', '
Figure_A8.pdf', 'Figure_A1.pdf', 'output', 'Figure_A15.pdf', 'Table_1.tex',
'Table_A8.tex', 'Figure_A12.pdf', 'Table_3a.tex', 'Figure_7.pdf', '
Figure_6.pdf', 'Table_A9.tex', 'Figure_A13.pdf', 'Figure_A17.pdf', '
Figure_5.pdf', 'Figure_A14.pdf', 'Figure_A5.pdf', 'Figure_A6.pdf']
Step 2: Executed 'query_vision_language_model(prompt='Find the sample with the
lower homicide rate per 100k in 2000 and the p-value from the predicted
distribution of endline trust in police.', input_image_paths=['results/
Figure_2.pdf', 'results/Figure_3.pdf'])'\n", " - **Reasoning:** \"There
are multiple figures in the results directory, and the file names do not
provide clear information about the figures' content.\"\n", " - **Status
:** 'error'\n", " - **Reason:** Error code: 400 - {'error': {'message': \"
You uploaded an unsupported image. Please make sure your image is below 20
MB in size and is of one the following formats: ['png', 'jpeg', 'gif', '
webp'].\", 'type': 'invalid_request_error', 'param': None, 'code': '
invalid_image_format'}}\n", " - **Error:** CommandExecutionError('Error
code: 400 - {\\"error\\\": {\\"message\\\": \"You uploaded an unsupported
image. Please make sure your image is below 20 MB in size and is of one the
following formats: [\\"png\\\", \\"jpeg\\\", \\"gif\\\", \\"webp\\\"].\", \\"
type\\\": \\"invalid_request_error\\\", \\"param\\\": None, \\"code\\\": \\"
invalid_image_format\\\"}}')
Step 3: Executed 'execute_shell(command_line='pdftoppm -png results/Figure_2.
pdf results/Figure_2 && pdftoppm -png results/Figure_3.pdf results/Figure_3
')'\n", " - **Reasoning:** \"The PDF files need to be converted to a
compatible image format (e.g., PNG) before querying the vision language
model.\"\n", " - **Status:** 'success'\n", " - **Output:** \n", "
\"'\n", "          STDOUT:\n", "          \n", "          STDERR:\n", "          \n", "
''
Step 4: Executed 'query_vision_language_model(prompt='Find the sample with the
lower homicide rate per 100k in 2000 and the p-value from the predicted
distribution of endline trust in police.', input_image_paths=['results/
Figure_2-1.png', 'results/Figure_3-1.png'])'\n", " - **Reasoning:** \"The
next step is to query the vision language model on the converted PNG images
to extract the required information.\"\n", " - **Status:** 'success'\n",
" - **Output:** \n", "          \"'\n", "          The homicide rate per 100k in
the year 2000 is shown in the first image. The solid line represents
Antioquia, and the dashed line represents Colombia as a whole. In 2000,
Antioquia has a higher homicide rate per 100k compared to Colombia.
Therefore, Colombia (the dashed line) has the lower homicide rate per 100k
in 2000.\n", "          \n", "          To find the p-value from the predicted
distribution of endline trust in police, we refer to the second image. The
relevant measure from the trust category is \"Change in trust in police (
endline - baseline).\" We look for the point estimate with the Block FE,
lagged DV (represented by triangles).\n", "          \n", "          Based on the
graph:\n", "          - The point estimate for the \"Change in trust in police
(endline - baseline)\" with Block FE and lagged DV crosses the zero line.\n",
"          - The corresponding confidence intervals also cross zero.\n", "
          \n", "          When the confidence intervals cross zero, it usually

```

```

indicates non-significance. In this plot, it suggests that the p-value is
above the typical threshold for statistical significance (usually 0.05).\n"
, "\n", "\n", "In summary:\n", "\n" - Colombia had a lower homicide
rate per 100k in 2000.\n", "\n" - The p-value for the predicted
distribution of endline trust in police, based on the provided graph for
the "\nChange in trust in police (endline - baseline)\n" with Block FE and
lagged DV, is not statistically significant (p-value > 0.05).\n", "\n"
'''

```

```

Step 5: Executed 'write_file(filename='report.json', contents='{\"fig From the
figure measuring homicide rate per 100k in the last 12 months, report the
name of the sample with the lower homicide rate per 100k in 2000.\": \"
Colombia\", \"fig From the figure measuring bootstrapped predictive
distribution of endline trust in police assuming mean regression at rate of
mean regression among unexposed citizens, report the p value from the
Heard of Meetings plot.\": \"> 0.05\"}')

```

D.3.2 Manually reproducing the code instead of using Docker (CORE-Bench-Medium)

On capsule-8234136, CORE-Agent with GPT-4o-mini tried to manually reproduce the code even when prompted to use Docker, causing the agent to eventually hit the context limit and fail. We found this problem to be more persistent on weaker models like GPT-4o-mini, which did not follow instructions as well as GPT-4o.

```

Step 1: Listed the contents of the current directory, resulting in a successful
output of 162 files and directories.
Step 2: Read the 'REPRODUCING.md' file successfully, which provided
instructions for reproducing results using a Docker image from Code Ocean's
registry.\n", "\n", "*"
Step 3: Executed 'pip install -r code/requirements.txt --no-input', resulting
in a failure during the installation of packages, specifically with numpy
due to metadata generation issues.
Step 4: Executed 'pip install numpy==1.18.5 --no-input', but the installation
failed due to a metadata generation error.
Step 5: Attempted to install 'matplotlib==3.2.2' to resolve dependency
conflicts, but the installation failed due to a missing FreeType dependency
.
Step 6: Installed 'libfreetype6-dev' and related packages successfully,
enabling Matplotlib to build.
Step 7: Reinstalled Matplotlib version 3.3.2 successfully after installing
FreeType.
Step 8: Executed 'main.py' with GOBI as the scheduler argument, resulting in a
success status but encountering a 'ModuleNotFoundError' for 'joblib'.

[...]

Step 65: Executed 'open_file(file_path='code/grapher.py')'\n", "\n" - **Reasoning
:** \"I need to adjust the import statements in 'grapher.py' to ensure it
can access the required modules directly, which will resolve the '
ModuleNotFoundError' for 'src'.\n", "\n" - **Status:** 'success'\n", "\n" - **
**Output:** File code/grapher.py has been opened and added to the context \
u2705
Step 66: Executed 'open_file(file_path='code/grapher.py')'\n", "\n" - **Reasoning
:** \"Next, I will adjust the import paths in 'grapher.py' to ensure it
references modules correctly.\n", "\n" - **Status:** 'success'\n", "\n" - **
Output:** File code/grapher.py has been opened and added to the context \
u2705
Step 67: Executed 'open_file(file_path='code/grapher.py')'\n", "\n" - **Reasoning
:** \"The next step is to modify the import statements in 'grapher.py' to
ensure it can access the necessary modules directly, which will resolve the

```

```

'ModuleNotFoundError' for 'src'.\ "\n", " - **Status:** 'success'\n", " -
**Output:** File code/grapher.py has been opened and added to the context
\u2705\n"
Step 68: Executed 'open_file(file_path='code/grapher.py')'

```

D.3.3 Being unable to install the correct version of dependencies (CORE-Bench-Hard)

In capsule-8807709, CORE-Agent with GPT-4o installed `network-diffusion` version 0.14.4. However, one of the import statements (from `network_diffusion` import `MultiSpreading`) threw an error because the import was only supported in version 0.6). The agent successfully realized it may need to install an older library version and performed a web search to see which version was applicable, but could not find the correct result within the cost constraint.

This example shows how reproducing a paper can be a difficult task, even for a human. Determining which library version to install without additional documentation can be hard without external knowledge.

```

Step 1: Executed 'read_file(filename='README.md')' and successfully identified
dependencies and installation instructions for 'epidemic.py'.
Step 2: Executed 'list_folder(folder='.')' and found: ['config.ini', 'LICENSE',
'README.md', '.gitignore', '.DS_Store', 'code/config.ini', 'code/epidemic
.py', 'code/LICENSE', 'code/gossip.ipynb', 'code/market_competition.ipynb',
'code/auxiliaries/model_real.png', 'code/auxiliaries/model_nd.png', 'code/
auxiliaries/florentine.mpx', 'code/auxiliaries/phase_3.png', 'code/
auxiliaries/phase_1.png', 'code/auxiliaries/phase_2.png', 'code/auxiliaries
/general_model.png', 'efficiency_tests/tests.ipynb', 'efficiency_tests/
data_ndlib.csv', 'efficiency_tests/time_measurements.py', 'efficiency_tests
/tested_functions.py', 'efficiency_tests/data_ndiff.csv', 'metadata/
metadata.yml'].
Step 3: Executed 'read_file(filename='code/epidemic.py')' and successfully
retrieved the content, including the import statements, configuration setup
, propagation model initialization, and node state settings for a multi-
spreading experiment.
Step 4: Executed 'pip install -r requirements.txt' - Result: ERROR: Could not
open requirements file: [Errno 2] No such file or directory: 'requirements.
txt'
Step 5: Executed 'pip install networkx numpy' and successfully installed
networkx-3.3 (numpy was already satisfied).
Step 6: Executed 'execute_shell(command_line='python3 code/epidemic.py')'
resulting in a 'ModuleNotFoundError: No module named 'network_diffusion''.
Step 7: Read the 'README.md' file successfully, revealing instructions for
using the 'network_diffusion' package, including setup via CodeOcean or a
Python environment, and details on running various example files.
Step 8: Executed 'web_search(query='python network_diffusion module
installation')' and found the relevant module 'network-diffusion' on PyPI
: https://pypi.org/project/network-diffusion/.
Step 9: Executed 'pip install network-diffusion' resulting in the successful
installation of 'network-diffusion-0.14.4' and its dependencies, but noted
a dependency conflict with 'numpy'.
Step 10: Executed 'python3 code/epidemic.py' resulting in an ImportError:
cannot import name 'MultiSpreading' from 'network_diffusion'.
Step 11: Executed 'pip show network-diffusion' and found that the module '
network_diffusion' is version 0.14.4, authored by Micha\u0142 Czuba and
Piotr Br\u00f3dka, licensed under GNU GPL v3, located at '/root/.cache/
pypoetry/virtualenvs/agpt-ttFEHEs8-py3.10/lib/python3.10/site-packages',
and requires 'dynetx', 'matplotlib', 'networkx', 'numpy', 'pandas', 'scipy',
and 'tqdm'.
Step 12: Executed 'ls /root/.cache/pypoetry/virtualenvs/agpt-ttFEHEs8-py3.10/
lib/python3.10/site-packages/network_diffusion' and found files: '__init__

```

```
py', '__pycache__', 'logger.py', 'mln', 'models', 'seeding', 'simulator.py',
', 'tests', 'tpn', 'utils.py'.
Step 13: Executed 'ls' command on 'network_diffusion/models' directory,
successfully listing files: '__init__.py', '__pycache__', 'base_model.py',
'dsaa_model.py', 'mic_model.py', 'mlt_model.py', 'tne_model.py', and 'utils
'.
Step 14: Executed 'grep 'class MultiSpreading'' in the specified directory; no
output was produced.
Step 15: Executed 'read_file(filename='code/epidemic.py')' successfully and
found the script initializes and performs a propagation experiment using
the 'MultiSpreading' class.
Step 16: Executed 'jupyter nbconvert --to notebook --execute code/gossip.ipynb'
with result: '/bin/sh: 1: jupyter: not found'.
Step 17: Reviewed 'README.md' successfully; detailed steps for running '
epidemic.py' and installing dependencies were found.
Step 18: Executed 'pip install ipykernel tqdm' successfully, installing
packages: ipykernel-6.29.5, comm-0.2.2, debugpy-1.8.5, ipython-8.26.0,
jupyter-client-8.6.2, jupyter-core-5.7.2, matplotlib-inline-0.1.7, nest-
asyncio-1.6.0, pyzmq-26.1.0, tornado-6.4.1, traitlets-5.14.3, jedi-0.19.1,
prompt-toolkit-3.0.47, stack-data-0.6.3, asttokens-2.4.1, executing-2.0.1,
pure-eval-0.2.3.
Step 19: Executed 'python3 code/epidemic.py' resulting in an ImportError:
cannot import name 'MultiSpreading' from 'network_diffusion'.
Step 20: Executed 'web_search(query='network_diffusion MultiSpreading class',
num_results=1)' successfully; found a relevant URL: https://network-
diffusion.readthedocs.io/en/latest/spreading_models.html.

[...]
```

D.3.4 Attempting to look up the capsule on CodeOcean (CORE-Bench-Hard)

Also in capsule-8807709, CORE-Agent with GPT-4o, after being unable to locate the requirements.txt file in the repository, attempted to look up the capsule on CodeOcean online. The agent ultimately did not succeed because JavaScript is required to render CodeOcean, which the agent did not have access to through its web browsing capabilities. However, this example highlights the care that agent developers must take during evaluation.

```
[...]

Step 60: Attempted to read 'requirements.txt' but encountered an error: [Errno
2] No such file or directory: '/home/crab/autogpt/environment/capsule
-8807709/requirements.txt'.\n", "\n", "*"
Step 61: Executed 'web_search(query='network diffusion site:github.com',
num_results=5)' and found relevant repositories, including \"GitHub -
GiulioRossetti/ndlib: Network Diffusion Library\" at https://github.com/
GiulioRossetti/ndlib.
Step 62: Executed 'web_search(query='codeocean capsule 8807709')' and found
links including the main capsule page [here](https://codeocean.com/capsule
/8807709) with additional details on compute capsules and their
functionalities.
Step 63: Executed 'web_search(query='https://codeocean.com/capsule/8807709')'
and retrieved multiple relevant URLs and excerpts related to Code Ocean
capsules.
Step 64: Executed 'web_search(query='https://codeocean.com/capsule/8807709')'
and found multiple relevant URLs, including https://codeocean.com/capsule
/8807709, which requires JavaScript to run.
Step 65: Executed 'web_search(query='https://codeocean.com/capsule/8807709')'
to find relevant information about Code Ocean's compute capsules;
```

```
successfully retrieved multiple URLs and excerpts detailing their features
and usage.
Step 66: Executed 'web_search(query='https://codeocean.com/capsule/8807709')'
with success, retrieving multiple relevant Code Ocean URLs and excerpts,
including https://codeocean.com/capsule/8807709 and https://codeocean.com/
capsule/8807709/tree/v3.
Step 67: Executed 'web_search(query='https://codeocean.com/capsule/8807709')'
and retrieved multiple relevant URLs, including https://codeocean.com/
capsule/8807709.
Step 68: Executed 'web_search(query='codeocean.com capsule 8807709',
num_results=1)'; found URL: https://codeocean.com/capsule/8807709.
Step 69: Executed 'write_file(filename='requirements.txt', contents='\\
nnetworkx\\nnumpy\\ntqdm\\nmatplotlib\\npandas\\nscipy\\nnetwork-
diffusion@https://github.com/anty-filidor/network_diffusion/archive/refs/
heads/master.zip#egg=network_diffusion')' successfully, resulting in the
file requirements.txt being written.
Step 70: Executed 'pip install -r requirements.txt' and successfully installed
most packages, but encountered an error with scipy due to missing BLAS/
LAPACK libraries.
Step 71: Executed 'execute_python_code(code='import network_diffusion\\nprint(
dir(network_diffusion))')' - Result: 'ModuleNotFoundError: No module named
'network_diffusion'.'.
Step 72: Ran 'python3 code/epidemic.py' and encountered an ImportError: cannot
import name 'MultiSpreading' from 'network_diffusion'.
Step 73: Executed 'execute_python_code(code='import network_diffusion;print(dir
(network_diffusion.models))')' resulting in 'ModuleNotFoundError: No module
named 'network_diffusion'.'.
Step 74: Executed 'execute_python_code(code='import network_diffusion; print(
dir(network_diffusion.models))')' resulting in 'ModuleNotFoundError: No
module named 'network_diffusion'.'.
Step 75: Executed 'import network_diffusion.models as models; print([cls for
cls in dir(models) if hasattr(getattr(models, cls), '__class__')])'
resulting in 'ModuleNotFoundError: No module named 'network_diffusion'.'.
Step 76: Executed 'web_search(query='network_diffusion MultiSpreading class
GitHub')' and found relevant resources, including a GitHub repository for
NDlib and various documentation and articles on network diffusion models.
Step 77: Executed 'web_search(query='MultiSpreading class site:github.com',
num_results=5)'; found five relevant GitHub pages discussing multithreading
in C++ and Java, but no direct references to the 'MultiSpreading' class.
Step 78: Executed 'web_search(query='CodeOcean account creation guide',
num_results=1)' and found "Create an Admin Account | Code Ocean VPC
Administration Guide" at https://docs.codeocean.com/admin-guide/deployment
-guide/create-admin-account.
Step 79: Executed 'read_file(filename='config.ini')' successfully, found '[
PATHS] output_dir: ./results'.
Step 80: Executed 'web_search(query='Create an Admin Account | Code Ocean VPC
Administration Guide', num_results=1)'\n", " - **Reasoning:** "Given our
progress and challenges, setting up CodeOcean seems vital to proceed
clearly and efficiently.\n", " - **Status:** 'success'\n", " - **Output
:** \n", "      '\n", "      ## Search results\n", "      ### "Create an
Admin Account | Code Ocean VPC Administration Guide"\n", "      **URL:**
https://docs.codeocean.com/admin-guide/deployment-guide/create-admin-
account \n", "      **Excerpt:** "Go to https://codeocean.[hosting domain
]/join to create an admin account : Before you invite users, we recommend
you to deploy a docker base image first. Visit Deploying Docker Images for
details. Learn to create the initial admin account on Code Ocean VPC.\n",
"      "'
```

[...]

E Reproducibility Study Details

We report the number of studies reproduced for each paper in Table 1 based on the format of the results provided by the papers’ authors:

- The following papers report the percentage of reproducible studies out of the total number of studies. We manually calculated the number of reproducible studies and rounded the result: Stockemer et al. (2018), Gertler et al. (2018), Collberg & Proebsting (2016), Hardwicke et al. (2021), Raff (2019).
- The following papers report the number of results or metrics that were computationally reproducible, rather than the number of papers: Gilbert et al. (2012), Trisovic et al. (2022), Samuel & Mietchen (2024), Pérignon et al. (2024), Belz et al. (2021).
- McCullough et al. (2006) reports an approximate number of papers reproduced. Authors state that they analyzed greater than 150 papers, with less than 15 replicated.

We manually analyzed the papers from the 2022 Machine Learning Reproducibility Challenge (Sinha et al., 2023). Of the 44 papers submitted to the challenge, 28 attempted to reproduce papers where both data and code were fully available. 10 of those 28 papers were only partially reproduced. We consider papers to be fully reproduced if all the main claims of the paper completely hold, even if the reproduced quantitative results slightly deviate from the original results. For example, we consider Livernoche & Sujaya (2023) a successful reproduction of the original paper because authors validate the original paper’s claims and results fall within the standard deviation reported in the original paper. On the other hand, we consider papers to have reproducibility errors if all the main claims of the paper cannot be reproduced, or if result values from the original paper deviate significantly from those of the reproduced paper. For example, we treat Brivio & Çöltekin (2023) as an unsuccessful reproduction because the highest accuracy score from the reproduced paper deviates significantly from the original paper although the original hypothesis was verified. We do not consider the results of additional experiments not contained in the original paper. Of the fully reproduced papers, many codebases contained errors, outdated packages, or limited documentation, requiring researchers to modify the codebase during the reproduction process.