ReplicatorBench: Benchmarking LLM Agents for Replicability in Social and Behavioral Sciences

Bang Nguyen*
University of Notre Dame
Computer Science and Engineering
Notre Dame, IN, United States
bnguyen5@nd.edu

Dominik Soós*
Old Dominion University
Computer Science
Norfolk, VA, United States
dsoos001@odu.edu

Qian Ma
Pennsylvania State University
Information Sciences and Technology
University Park, PA, United States
qfm5033@psu.edu

Rochana R. Obadage
Old Dominion University
Computer Science
Norfolk, VA, United States
oruma001@odu.edu

Zack Ranjan
Pennsylvania State University
Information Sciences and Technology
University Park, PA, United States
sjr6223@psu.edu

Sai Koneru
Pennsylvania State University
Information Sciences and Technology
University Park, PA, United States
sdk96@psu.edu

Timothy M. Errington
Center for Open Science
Washington, DC, United States
tim@cos.io

Shakhlo Nematova
Center for Open Science
Washington, DC, United States
shakhlo@cos.io

Sarah Rajtmajer
Pennsylvania State University
Information Sciences and Technology
University Park, PA, United States
smr48@psu.edu

Jian Wu
Old Dominion University
Computer Science
Norfolk, VA, United States
jwu@cs.odu.edu

Meng Jiang
University of Notre Dame
Computer Science and Engineering
Notre Dame, IN, United States
mjiang2@nd.edu

Abstract

The literature has witnessed an emerging interest in developing and evaluating AI agents for automated assessment of research claims in scientific papers. Existing benchmarks focus primarily on the computational aspect of this task, testing agents' ability to reproduce or replicate research outcomes when having access to the code and data. This setting, while foundational, (1) fails to capture the inconsistent availability of new data for replication as opposed to reproduction, and (2) lacks ground-truth diversity by focusing exclusively on fully reproducible or replicable papers, thereby failing to evaluate an agent's ability to identify non-replicable research. Furthermore, most benchmarks only evaluate the final reproducibility or replicability outcomes without an evaluation of the process. In response, we introduce ReplicatorBench, an end-to-end benchmark, including human-verified replicable and non-replicable research claims in social and behavioral sciences, for evaluating AI agents in research replication across three stages: (1) extraction of relevant information and retrieval of replication data; (2) design and execution of computational experiments; and (3) interpretation of replication results, allowing a test of AI agents' capability to mimic the activities of human replicators in real world. To set a baseline of AI agents' capability, we develop ReplicatorAgent, an agentic framework equipped with necessary tools like web search and iterative interaction with sandboxed environments, to accomplish tasks in ReplicatorBench. We evaluate ReplicatorAgent across four underlying large language models (LLMs), as well as different design choices of programming language and levels of code access. Our findings reveal that while current LLM agents are capable of effectively designing and executing computational experiments, they struggle with retrieving new data, necessary to replicate a claim. All code and data are publicly available at: https://github.com/CenterForOpenScience/llm-benchmarking.

Keywords

Research replication, LLM agents, Computational social science

Resource Availability:
The prompts, structured output schemas, and supplementary materials associated with this paper are publicly available in an archival Zenodo artifact at https://doi.org/10.5281/zenodo.20506946.

1 Introduction

Recent advances in large language models (LLMs) have enabled autonomous agents that can read documents, write and execute code, use tools, and operate in sandboxed environments [18, 22]. These agents have been adopted into various stages of the research lifecycle, from idea creation, experimental design, and execution to results synthesis and paper writing [8, 9, 13]. In social sciences, there has been an increasing interest in the development and testing of LLM agents for research reproduction and replication [5, 14].

*The authors contributed equally to this research.

,

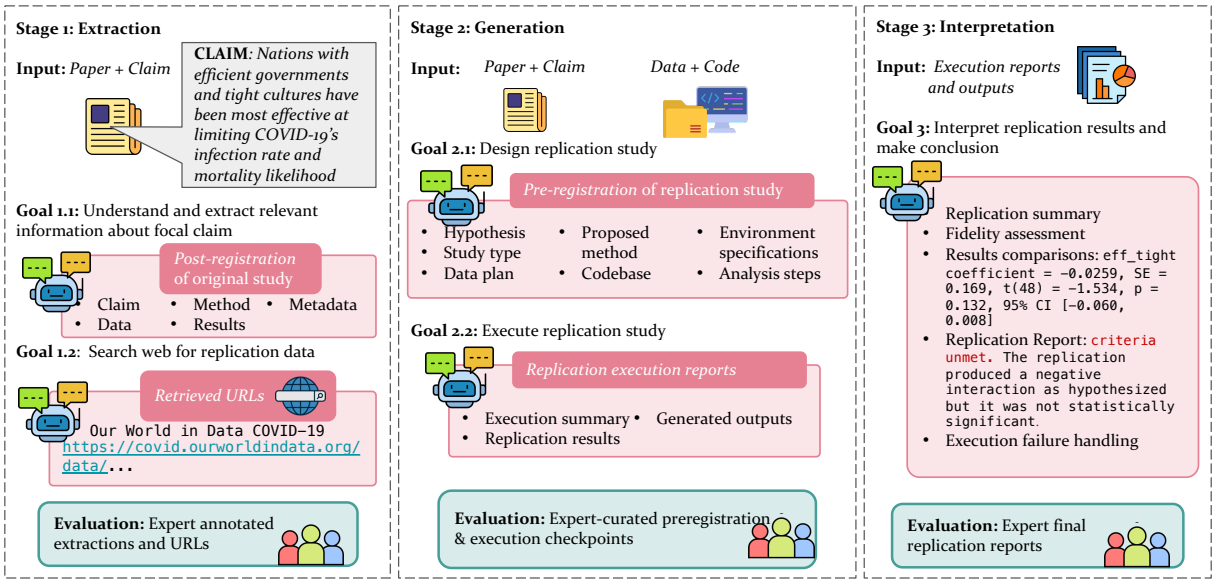

Figure 1: Given a research paper and a focal claim, ReplicatorBench decomposes the replication process into three stages. (1) *Extraction* assesses the agent's ability to gather relevant information about the claim and retrieve data resources for replication; (2) *Generation* assesses the computational capacity of an agent to generate and execute code for replication; (3) *Interpretation* assesses the agent's ability to interpret computational outputs and make conclusions regarding the focal claim's replicability.

While research *reproduction* attempts to confirm whether the same findings can be obtained using the original data samples and methods, *replication* investigates the generalizability of methods when applied to new data samples [11, 12]. These tasks, while essential to academic and research integrity, are resource-intensive, providing a promising use case for autonomous LLM agents.

Several benchmarks have been proposed to develop and evaluate LLM agents on research reproduction tasks, including CORE-Bench [14], PaperBench [15], and REPRO-Bench [5]. LLM agents have the potential to assess the reproducibility or replicability of research claims in a controlled setting. By definition, replication requires researchers to locate or construct a new data sample [11]. Existing benchmarks operate under the assumption that the new data sample is readily available to the agent [5, 14, 15]. There are no benchmarks designed to assess agents' capability to replicate research claims in a setting where a new data sample must be retrieved in advance. To avoid error propagation in later stages, the new-sample data are curated and packaged for the agent rather than collected live from the open web. In addition, most existing benchmarks focus on evaluating the final reproducibility or replicability outcome without considering the human replicators' process.

To fill these gaps, we introduce ReplicatorBench, a benchmark for replicating published research claims in social and behavioral sciences (SBS). The benchmark includes 39 instances, each corresponding to a paper with a human expert replication report produced in the SCORE project [1]. SCORE (Systemizing Confidence in Open Research and Evidence) is a DARPA program, in partnership with the Center for Open Science, that coordinated a large-scale extraction of social-behavioral claims and human reproduction and replication efforts to assess each claim's credibility [1].

In SCORE, replication studies are documented as an end-to-end process rather than a single report, starting from preregistering the replication plan with criteria for replicability, Internet-retrieved data sources, and proposed methodology, to conducting the replication study with documented code and data, and finally creating a report to assess the claim's replicability. This process-oriented documentation allows evaluation of the agent's behavior at each stage of a replication study with a hierarchy of checkpoints rather than only the end outcome [19]. Here, a checkpoint is a point during the replication process at which a result is produced and evaluated.

The agent replication outcome of each instance in ReplicatorBench falls into one of two categories: (1) *criteria met*, where the preregistered criteria are satisfied on the replication sample, or (2) *criteria unmet* otherwise. This binary setting is aligned with human replication results and adopted by existing replicability benchmarks. To evaluate agent performance, we measure the alignment between the agent's concluded outcome and the ground-truth human outcome provided by the SCORE project. ReplicatorBench supports fine-granular evaluation and gives AI agents partial credits through evaluating 3, 128 gradable checkpoints across three main stages of 39 instances: Extraction, Generation, and Interpretation (illustrated in Figure 1 and detailed in §3.1).

As a baseline study, we develop ReplicatorAgent, a tool-using replication agentic framework capable of inspecting directories, reading and editing files with minimal diffs, resolving dependencies, and rerunning analyses until it produces interpretable outputs or reaches a well-justified conclusion (criteria met or unmet). We also study a practical trade-off for replication agents: executing native non-Python codes (e.g., R) provided by the original authors or human replicators versus executing Python codes translated from non-Python codes. We found that this practice streamlines the replication process by reducing installations of language compilers and software dependencies, which resulted in fewer execution failures. The trade-off is that it may introduce information loss that

produces incorrect outputs, which in turn negatively affects the reliability of the replicability assessment.

Our contributions are threefold.

(1) We introduce ReplicatorBench, a novel benchmark consisting of 39 instances targeting evaluating AI agents' capability of replicating research claims in scientific literature in social and behavioral sciences, mimicking the real-world setting in which the new data for replication is not given. In addition to the binary outcome, the benchmark breaks the replication process into three stages, which encapsulate a total of 3,128 gradable checkpoints for partial credits.
(2) We develop ReplicatorAgent, a tool-using agentic framework and a sandbox orchestration setup that supports iterative debugging and end-to-end replication attempts.
(3) We evaluate ReplicatorAgent against ReplicatorBench under several configurations, such as the choice of LLMs and whether to translate native codes to Python. We highlight where today's agents succeed and fail, and the root causes of failures in the replication process, and shed light on the real-world replication challenges.

2 Related Work

2.1 Research Reproducibility and Replicability

Existing research has revealed significant evidence regarding the lack of replicability in fields such as psychology [4] and economics [3]. Authors have explored the connections (or lack thereof) between replicability and factors such as authors' experience, prestige, and the availability of data and code [2]. Recent studies have also acknowledged the time- and resource-consuming nature of manual replication and motivated the need for automated methods to replicate research claims [1]. ReplicationBench [23] offers a principled benchmark to test and evaluate such methods.

2.2 Benchmarking LLM Agents for Assessing Reproducibility and Replicability

Existing literature on this subject predominantly revolves around benchmarking LLMs' computational capacity for assessing reproducibility and replicability. For example, LMR-BENCH [21] withholds key implementation components from language modeling papers and evaluates whether LLMs can reconstruct them. Additionally, LLMs have also been evaluated on their ability to solve coding problems in machine learning research in ResearchCodeBench [6], in natural sciences in SciCode [16], and in natural language processing in SciReplicate-Bench [20]. Benchmarks like CoreBench [14], PaperBench [15], and ReplicationBench [23] investigate autonomous computational reproduction or replication, testing self-driven AI agents continuously from code implementation to execution and debugging without human intervention.

Replicating research claims requires mapping a focal claim to a prespecified (statistical) criterion to determine whether that criterion is met in a new dataset, using the same analytical methods. CORE-Bench [14], a benchmark dataset consisting of 90 papers in Computer Science, Biology, and Social Sciences, addresses this requirement by introducing a subtask of interpreting execution outputs. However, CORE-Bench only includes fully reproducible claims, and therefore, cannot be used for evaluating the agents' capability for irreproducible claims. REPROBench [5] is a benchmark consisting of 112 papers in social science. Although papers in this benchmark include different levels of reproducibility to the test set, the benchmark situates the reproduction task within a controlled setting where agents conveniently have access to both code and data. Our ReplicatorBench addresses this gap by introducing a resource retrieval task as a first step to the challenge commonly faced by researchers during replication. Furthermore, our ReplicatorBench differs from most existing benchmarks by evaluating the replication process, starting from information extraction from papers until interpreting computational results to draw conclusions. It is the first replication benchmark that provides fine-grained checkpoints to score the AI agent's capability to replicate research claims in social and behavioral sciences. Table 1 provides a comparison between existing benchmarks and our work.

3 ReplicatorBench

3.1 Task Design

ReplicatorBench defines replication tasks for a research paper by mimicking human experts' replication process. The goal of a replication study is to assess whether the focal claim can be verified when tested on a new, similar dataset obtained independently from the dataset used in the research paper. We decompose this workflow into three stages: (1) **Extraction**, in which the agent "reads" the manuscript and extracts relevant information, such as method description and evidence, based on the focal claim, and then retrieve new data from the Internet for replication; (2) **Generation**, in which the agent prepares the replication by completing a preregistration of the replication plan, setting up an executable environment, executing the replication codes, and producing the intermediate results; the results of this stage include intermediate and final raw output that is ready to be further analyzed and interpreted in the third stage; and (3) **Interpretation**, in which the agent locates the agent's replication output, extracts relevant results, derives directly comparable results (if needed), compares them against the preregistered criteria and draw the conclusions.

3.1.1 Extraction. Given the focal claim and the original paper in a PDF format, the agent is expected to perform two subtasks in this stage. First, the replication process starts with gathering key information that could be used in the replication process about the claim. Particularly, the agent is tasked with producing a structured document with extractions of 24 gradable information pieces surrounding *five* dimensions: *focal claim*, *data*, *method*, *results*, and *metadata*. We refer to this document as the *post-registration* of the *original study*. Next, the agent is tasked with searching for the replication data sample. In our benchmark, we model this process as an information retrieval task by requiring an agent to identify the appropriate replication data from the Web. The output of the extraction stage contains the *post-registration* document and a list of URLs for collecting replication data.

3.1.2 Generation. This stage addresses the computational aspect of the replication process that focuses on data processing and modeling. When humans replicate research claims, they often encounter a situation in which they do not have access to the original code or

Feature	CORE-Bench	PaperBench	REPROBench	ReplicatorBench (Ours)
Domain	Computer science, Biology, and Social sciences	Machine Learning	Social and Behavorial Sciences	Social and Behavorial Sciences
Paper Source	CodeOcean	ICML 2024	Journals, preprints, Twitter/X, Retraction Watch	Peer-reviewed journals
# Papers	90	20	112	39
Nature of Work	Reproducibility	Reproducibility	Reproducibility	Replicability
Task design & evaluation	Outcome-oriented	Outcome-oriented	Outcome-oriented	Both process- and outcome-oriented
Labels	Single label: reproducible	Single label: reproducible	4 reproducibility levels (4 levels from *irreproducible* to *fully reproducible*)	Binary (*criteria met* and *criteria unmet*)
Input	Code, original data, paper	Paper	Paper, original data, code, documentations, findings list	Paper, focal claim
# Tasks	270	8,316	112	3,128
Evaluation	Numerical range check	LLM scorer, hierarchical aggregation	Comparison against manual ground truth	3-stage aggregated scores (Auto + LLM)
Evaluated LLMs	GPT–4o, GPT–4o–mini	O3–mini–high, GPT–4o, Gemini–2.0, DeepSeek–R1, o1, Claude–3.5	GPT–4o	GPT–4o, GPT–5, GPT–5–mini, o3
Internet access	Yes	Yes	No	Yes

Table 1: Comparison of reproducibility benchmarks.

the original code was not executable due to compatibility or other reasons. To mimic this situation, we design an *easy* setting in which both replication data and the code are available, and a *hard* setting in which only the replication data is available. Given either setting, we further break this stage into three steps.

Step 1: Design. Mimicking the preregistration by human replicators, the agent is required to *preregister* the research plan before carrying out the replication. This step involves documentation of hypotheses, data, and method planning, as well as preparation of computational environment. Under the easy setting, this step sets up the virtual environment by installing the chosen programming language and necessary software dependencies, as well as resolving visible issues such as non-supported programming scripts or filename inconsistencies. Under the hard setting, the agent is expected to generate code following the original study's methodology description. The output of this step is a structured *preregistration* document with 30 gradable information pieces that reflects the plans for the replication across *seven* dimensions: *hypothesis*, *study type*, *data plan*, *proposed methodology*, *codebase*, *environment specifications*, and *analysis steps*.

Step 2: Execution. In this step, the agent performs the replication plan made in the last step. This includes processing the replication data, building models, deriving intermediate results, and generating raw results that could be later used for interpretation. This step also includes iteratively debugging software issues at compilation and run time toward a successful execution of the code (either the original or re-implemented). The output of this step is a structured document that reports how well the replication study is aligned with its preregistration, focusing on execution summary, intermediate results, and non-textual outputs such as tables and figures.

Economics	10	Political Science	4
Education	2	Psychology & Cognitive Science	8
Public Administration	1	Health	9
Sociology	5		

(a) Topics of 39 Papers.

Extract		Generate		Interpret	Outcome	
Info	Web	Design	Exec.		Criteria Met	Unmet
936	86	1170	390	507	20	19

(b) Distributions of 3,128 Tasks.

Table 2: Statistics of ReplicatorBench. All tasks were created from the entire sample of 39 papers, with the exception of the Extract-Web substage, which applies only to the 19 observational studies whose replication data is available on the web.

3.1.3 Interpretation. The goal of the Interpretation stage is to assess the outcome of a replication attempt, taking as input the execution output and logs, along with all inputs and outputs associated with the previous stages. First, the agent inspects the execution logs to checks what was executed versus what was planned in the design step. Next, it compares the final results of the execution step to the original study results recorded in the post-registration; verifying key reported statistics and evaluating the hypothesis tied to the focal claim. Third, the agent is required to determine whether the inference criteria for the focal claim are satisfied (i.e., a statistically significant effect ($\alpha = 0.05$, two-tailed) in the same pattern as the original study) and draw the final conclusion. The output of

this stage is a structured document with 13 gradable result interpretations across *seven* dimensions: *summary*, *fidelity assessment*, *results comparisons*, *replication report*, and *execution failure handling*. We provide an overview of the task templates and supplementary artifacts in Appendix C.

3.2 Rules

Through the replication workflow, an agent has read and write access to all files in the designated workspace on a dedicated computer. The agent has the permission to install, uninstall, and execute software packages. The agent works on the replication task in a closed environment, except for the Extraction stage, in which they are provided with Internet access. ReplicatorBench technically does not require any human intervention. However, as a precaution, we implemented an optional human-check function for all executable commands and file-writing actions. When enabled, actions are executed only after approval by a human supervisor. We observed no significant unsafe actions when testing the agent.

4 Dataset

ReplicatorBench evaluates LLM agents on replication tasks over a set of focal claims, each identified by domain experts from the papers compiled by the SCORE project [1]. These papers are sampled from peer-reviewed journals in six subject categories in social and behavioral sciences [1]. From the broader set of SCORE replications, we keep only papers where (1) the focal claim can be tested with a single quantitative check and a clear pass/fail criteria, and (2) the core materials needed for replication are available (preregistration/replication plan, data access, and code documentation). We do not restrict the programming languages used in the replication code. Since many SCORE cases are missing at least one of these pieces, this filtering yields 39 studies. Of these, 20 are observational in which the replication sample can be built from online or archival sources, and 19 are experimental studies in which the replication team must collect data through controlled experiments.

For each paper, a team of human researchers conducts a replication study to assess the replicability of the focal claim using a new dataset. The human replicators first pre-register an analysis that specifies data construction, models, and decision criteria used to judge whether the claim replicates. For each claim, the human replication team delivers a replication report documenting the preregistered plan, how the new data were obtained and processed, instructions to execute scripts, intermediate results, and the final conclusion based on whether the *preregistered criteria for a successful replication are met* (e.g., an effect that is statistically significant and in the same direction as the original study). These reports provide the ground truth for evaluating the agents. By breaking down the replication workflow of each claim into three stages, the benchmark contains 3,128 task instances (Table 2).

4.1 Grading

We pair each stage (see §3.1) with manual documentation from a human replication effort to evaluate the performance of the agent.

Reference for Extraction. To assess the quality of the agent's extracted information, we recruit three human annotators, all with research experience, to annotate references for the extraction tasks based on the original paper and focal claim. We ensured all three annotators passed a test to annotate relevant information from a representative paper before performing a complete annotation of the 39 papers. We adopt a leave-one-out evaluation setting where we use extracted information from two annotators as the ground-truth reference and the extracted information by the third annotator for computing a baseline performance for the extraction stage.

The ground truth of the data retrieval task is the URLs linking to the new data documented by human replicators in the preregistration document. Agents are not provided any candidate URLs. Given the original paper and the claim to replicate, an agent must perform an open web search to locate the appropriate data source.

Reference for Generation-Design. To assess the quality of the agent's preregistered plan for the replication study, we also use the original preregistration document by the human researchers. Specifically, we rely on the entire preregistration document, which include reported data cleaning, preprocessing procedures, and proposed methodologies in a single PDF or DOC file, to measure how well-developed the agent's own plans are.

Generation-Execution Checkpoints. As the generation stage is mainly concerned with the computational capabilities of the agent, we also identify 10 reference-free checkpoints for each claim on which the agent is evaluated in this stage. The *computational design* checkpoints include setting up the environment, identifying package dependencies, and investigating the file system; the *execution* checkpoints assess code execution and debugging capabilities, as well as appropriate documentation of execution outputs for the interpretation stage. The checkpoint schema used for execution evaliation is included in the supplementary artifact [10].

Reference for Interpretation. The final report produced by the human replication team, in a single PDF or DOC file, is used as a reference for the Interpretation stage.

5 Evaluation Metrics

Despite requiring agents to format their responses in a structured template, LLM-generated texts remain open-ended and are sensitive to reference-based metrics like BERTScore or ROUGE [7, 24]. We compare the effectiveness of LLM-as-a-judge (*LLMEval*) [25] with other text-generation metrics in Appendix B and found that LLMEval can provide reliable judgment of agent's replication attempts against the reference in our benchmark. Thus, we adopt LLMEval for automatic evaluation of extracted and generated results in all checkpoints as follows:

- *Extraction*: For each piece of information, the LLM judge assigns a score in a 0-3 range to measure how well extracted information matches the reference, with 0 indicating a poor match with the reference and 3 indicating a perfect match. The final score of this stage is calculated as the macro average of all 24 extracted fields, distributed across the five dimensions discussed in §3.1.
- *Generation - Design*: The LLM judge directly uses the human-submitted preregistration document, in a PDF or DOC format, to evaluate the quality of the agent-generated preregistered plans. For each field in the generated plan, the judge assigns a score of 0 or 3, with 0 indicating a poor match with the reference and 3 indicating a perfect match. The final score of this stage is

calculated as the macro average of 30 generated fields, distributed across seven dimensions discussed in §3.1.

- *Generation - Execution*: The LLM judge inspects the logs and execution outputs in the design and execution steps to determine whether the agent has addressed all 10 generation checkpoints. The LLMEval assigns a score of 0 or 1 for each generation checkpoint, with 0 indicating an unsatisfactory attempt while 1 indicates a satisfactory attempt. The final score of this stage is calculated as the macro average of 10 checkpoints across two dimensions *computational design* and *execution*.
- *Interpretation*: The LLM judge compares the agent's interpretation directly against the human replication report. For each checkpoint instance in this stage (e.g., statistical results and overall conclusion), the LLMEval assigns a score in a range of 0 to 3 for semantic match, with 0 indicating a poor match with the reference and 3 indicating a perfect match. The final score of this stage is calculated as the macro average of 13 generated fields, distributed across seven dimensions discussed in §3.1.

In addition to reporting rubric-based scores with LLMEval, we also report binary classification metrics for the final checkpoint in which we compare the agent's conclusion using the human researchers' conclusion (i.e., whether they both agree on *Criteria Met* or *Criteria Unmet*). We observe that it is possible that the agent makes an inconclusive decision when it is unable to execute the code successfully or when it fails to explore the execution outputs during the interpretation stage. We treat such decisions as incorrect decisions. We report precision, recall, and F1 scores for this checkpoint. Our benchmark contains 20 human replication studies labeled as *criteria met* and 19 labeled as *criteria unmet*. To treat these two classes equally, we use macro aggregation to reflect agents' ability to assess both replicable and non-replicable claims.

For the data search task in the Extraction stage, we evaluate the retrieved URLs with alias matching. Because resources may have multiple entry points and landing pages that require human expert authentication, we treat the model's output as correct if it matches any of the aliases. We report macro precision, recall, F1, hit@*any*, and hit@*all*, with both micro and macro aggregation across cases. Hit@*any* measures the frequency with which the search model found at least one required resource from the gold set, and hit@*all* measures if all required resources were found.

6 Experimental Setup

As a baseline study, we developed an agentic framework called ReplicatorAgent to demonstrate the performance of frontier LLMs against our benchmark.

6.1 ReplicatorAgent

ReplicatorAgent follows a ReAct-style loop [22], in which it alternates between reasoning and actions to execute the replication workflow and produce the required artifacts in each stage, such as the preregistration and post-registration described in §4, and execution logs. One key design consideration is to support iterative debugging rather than treating the first execution failure as terminal (Figure 2). We include an example execution log showing this iterative debugging in practice in Figure 5.

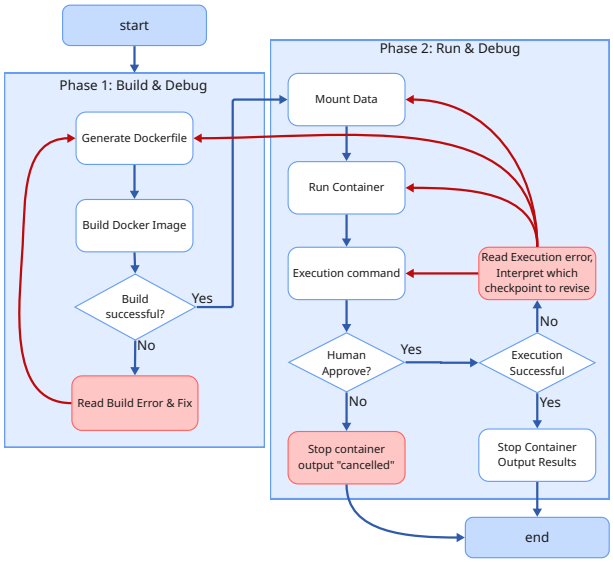

Figure 2: The Generation-Execution stage has two main phases: build and debug, and run and debug. We developed iterative debugging to fix issues and improve performance.

The ReplicatorAgent is provided with a simple palette, which was motivated by the commonly experienced replication failures we experienced. This palette is effective in reducing single-point failures such as hard-coded paths, missing dependencies, and partial outputs. The palette includes file and directory inspection, targeted file readers, dataset inspection utilities such as column names and basic summaries, file editing that produces git-style diffs for localized patches, and constrained rewrites that require explicit overwrite flags to reduce destructive edits.

6.2 Execution Environment

We run the agentic framework in a sandboxed container environment orchestrated by a runner, a lightweight orchestration script that builds an image, mounts the code and data into a standard directory layout, executes the analysis entrypoint, and exports the logs and other artifacts produced. Sandboxing the environment serves as a safety measure and ensures that our results are reproducible by recording the exact environment and executed commands.

As agents write and execute code in the environment, raw execution logs may exceed context limits and alter the statistics needed for the final criteria decision. In these cases, we truncate the logs by keeping the first 2000 lines and then prompt the agent to rewrite the analysis entrypoint to produce a compact summary and rerun the analytics so that key information prompted early on, such as the preregistered coefficients and p-values are included within the context limit. This practice produces compact outputs that are used by the *Interpretation* stage without fabricating missing evidence.

6.3 Design Choices

Native vs. Python Modes. Inspired by existing findings regarding LLMs' preference for Python during code generation and execution

[17] and thereby superior performance in Python coding compared to other languages, e.g., R [14], we evaluate two execution configurations: *Native* mode, in which the agent runs the replication package in its original programming language when possible and is explicitly instructed not to translate code; and *Python* mode, in which the agent is required to translate a non-Python script into Python prior to setting up the execution environment.

Data-only Setting. Inspired by the real-world challenge, we test the agent under two possible settings, depending on whether the code is accessible in the replication package. In both settings, the agent has access to the new data (the agent is still required to accomplish the data retrieval task). However, in the data-only setting, the agent does not have access to the native codes. To accomplish the replication tasks, the agent must generate its own implementation of the methods based on the algorithms conveyed in the paper. This ablation study tests the agent's capability to implement methods solely by reading documentation and extracted specifications.

Underlying LLMs. We evaluate our framework by instantiating its policy with multiple frontier LLMs. Specifically, we experimented with GPT-4o, GPT-5, GPT-5-mini, and o3, which differ in reasoning strength and tool use behavior. All models are evaluated under the same agent architecture, tools, and execution settings to draw a fair comparison.

7 Results

7.1 Main Results

Table 3 presents the performance of ReplicatorAgent, initiated with different LLMs, in the three stages of ReplicatorBench in *Python mode,* where all non-Python scripts are translated before execution to keep the environment consistent across models. We report the *Native* vs. *Python* comparison separately. This control for differences in language-specific runtimes helps isolate the LLM's bias towards certain programming languages.

7.1.1 Extracting Claim Information. Under leave-one-out evaluation, our human annotators achieve an LLMEVal score of 71.33 on information extraction. GPT-5 achieves the highest score (63.65) and GPT-4o achieves the lowest score (57.56) for this task. All evaluated LLMs have yet to measure up to human performance in filling out the post-registration.

7.1.2 Web search for replication data. Because of the noticeable performance difference between the data search and other tasks, we compared base models and several fine-tuned models on the data retrieval search task. The results in Table 4 indicate that among fine-tuned search models, `o3-deep-research` achieves the strongest overall performance and the highest coverage while `gpt-4o-search-preview` has higher hit@*any* than `gpt-5-search-api` , and `gpt-5-search-api` is higher on hit@*all*. Interestingly, tool-augmented base models exhibit substantially higher coverage overall , but this comes with differing precision–recall tradeoffs: `GPT-4o` has the highest precision and F1 , whereas GPT–5 achieves higher recall and the best hit rates despite lower precision. Base models benefit from stronger long-context reasoning and iterative query refinement, which are critical for mapping research claims to dataset sources, whereas search-tuned models may optimize generic web relevance and thus return plausible but non-canonical pages that are penalized under domain-based grading.

7.1.3 Computational experimentation. The evaluation provided by LLMEval indicates that o3 and GPT–5 have the strongest computational performance for the execution stage. However, o3 exhibits a better performance during the preceding design stage that plans for the replication study compared to GPT–5. Upon manual inspection, we found that GPT–5 tends to shift effort from design to the execution stage. For example, the model only generates planning steps for the translation of the given code into Python during the *design* step and then actually performs the data inspection and code translation in the *execution* step. This difference in where the translation work is surfaced (Design vs. Execute) aligns with the stage-wise LLMEval rubric: designs that are more explicit and immediately actionable score higher, which explains why o3 achieves a better Design score than GPT–5. This is not a substantive failure of GPT–5, but rather a different task decomposition strategy.

7.1.4 Results interpretation and replication outcomes. Given an effective performance during the execution stage, our GPT–5 and o3 agents arrive at conclusions in the Interpretation stage with LLMEval scores of 91.26 and 94.54, respectively, compared to 82.25 and 87.02 by GPT–4o, and GPT–5–mini. Regarding binary classification performance on replication outcome, GPT–5 achieves the best performance across all LLMs, with a 10% net gain compared to the runner-up o3. GPT–5–mini has yet to demonstrate competence for this task, only achieving a Macro F1 of 47.86.

7.2 Additional Analyses

7.2.1 Native vs Python. In this analysis (see Figure 3), we highlight that an agent's capability for computational experimentation may not necessarily correlate with their capability to sensibly arrive at the correct replication outcome. Results indicate that Python mode often improves the agent's capability to reach an executable run when the native codes fail due to dependency installation or environment compatibility issues. Although this practice increases the performance of the execution stage, it does not necessarily guarantee the reliability of the final assessment regarding the focal claim's replicability. We attribute this phenomenon to potential hallucinations and information loss as the agent translates original code into Python. Incorrect code translations may lead to different numerical outputs compared to the execution of the original code. These errors are then propagated to the interpretation stage. We provide an example of such errors in Figure 6 in Appendix A.2. Figure 3 highlights the results for GPT-4o in this analysis, and we observe similar trends in GPT-5 and o3 (see Table 8 in Appendix B.3).

7.2.2 Data-only Setting. Figure 4 compares the performance of GPT-5 ReplicatorAgent with and without access to human-written replication code. We observe an improvement in preregistering the research plan (higher LLMEval score for the design stage) when the agent has to write and execute code on its own. Under this setting, the agent typically spends more turns exploring the data to help generate the replication code, in turn providing a more detailed data plan for the preregistration. However, as the agent has to navigate both computational performance (generating and executing codes) and fidelity to original methodology in the generated code,

Model	Extract						Generate		Interpret	Outcome			
	Info	*Web*					*Design*	*Execute*					
	LLMEval	Prec.	Rec.	F1	hit@*any*	hit@*all*	LLMEval	LLMEval	LLMEval	Acc.	Prec.	Rec.	F1
o3	60.96	13.59	28.60	16.68	57.89	15.79	**83.82**	**95.18**	**94.54**	66.67	67.06	67.07	66.67
GPT–4o	57.56	**21.75**	22.53	**19.49**	57.89	10.53	77.45	80.48	82.25	61.54	61.45	61.51	61.44
GPT–5–mini	62.93	7.09	28.81	10.56	57.89	**15.79**	69.20	69.07	87.02	48.72	50.50	50.50	47.86
GPT–5	**63.65**	7.41	**30.62**	10.95	**63.16**	**15.79**	79.38	95.14	91.26	**76.92**	**78.53**	**77.78**	**76.86**

Table 3: Performance of our LLM agents across different stages against ReplicatorBench in Python mode. GPT-4o is used as the backbone model for LLMEval. All evaluation scores are normalized to a 0-100 scale. Macro aggregations are used.

Setting	Model	Macro P	Macro R	Macro F1	Micro P	Micro R	Micro F1	Hit@*any*	Hit@*all*
Search-tuned	o3-deep-research	**25.35**	22.09	**23.26**	18.57	15.85	17.11	52.63	10.53
	gpt-5-search-api	7.19	11.58	7.82	10.34	3.66	5.41	15.79	10.53
	gpt-4o-search-preview	9.56	11.21	8.82	9.21	8.54	8.86	26.32	5.26
Base + tool	gpt-4o	21.75	22.53	19.49	**21.54**	17.07	**19.05**	57.89	10.53
	gpt-5	7.41	**30.62**	10.95	6.88	**23.17**	10.61	**63.16**	**15.79**
	gpt-5-mini	7.09	28.81	10.56	6.90	17.07	9.82	57.89	**15.79**
	o3	13.59	28.60	16.68	12.39	17.07	14.36	57.89	**15.79**

Table 4: Web search subtask performance across search-tuned models and base models equipped with web search tools. We report macro and micro precision (P), recall (R), and F1, along with hit@*any* and hit@*all*. All values are scaled to a 0–100 range.

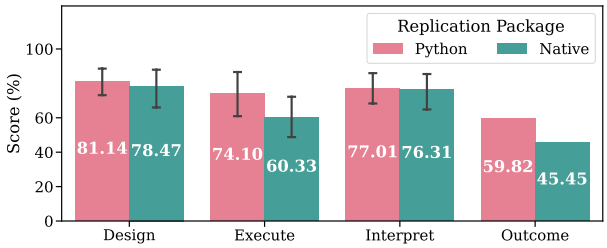

Figure 3: Performance of GPT-4o ReplicatorAgent in a Python-only setting compared to Native setting. Bars show LLMEval stage scores with error bars calculated as a 95% confidence interval of the mean. Outcome reports macro F1.

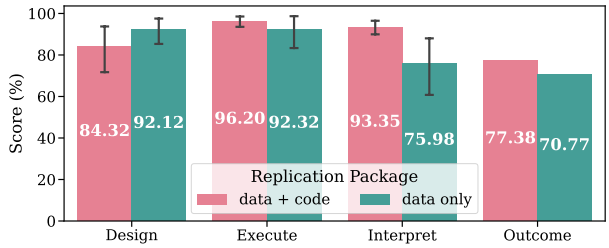

Figure 4: GPT-5 ReplicatorAgent with data+code versus data-only access. Bars show LLMEval stage scores with error bars calculated as a 95% confidence interval of the mean. Outcome reports macro F1.

we observe a drop in the agent's alignment with human researchers in their final assessment of the claim's replicability (lower scores for interpretation stage and replication outcome).

7.3 Error Categories

To support error analysis, we define the following error categories and use them to label errors when the replication is *Partial Success* or *Failure*. A single run may be assigned multiple error labels.

- *Setup errors.* The run cannot be started due to environment or dependency problems, e.g., installation errors, or missing files.
- *Input data errors.* The dataset cannot be loaded or does not match what the analysis expects, e.g., corrupted inputs, missing required variables, or incompatible formatting.
- *Implementation errors.* The run does not execute the intended procedure, e.g., omitting a required variable, incompatible variable types, or implementing a different procedure.
- *Result extraction errors.* The run does not produce the required outputs, e.g., target quantities cannot be located, or expected output files are missing.
- *Timeout errors.* The run does not finish within the allowed time, or the agent reaches its iteration limit.

We analyzed the distribution of error types across all models to identify potential bottlenecks in replication (see Appendix A for details). Our analysis reveals that Input data errors, specifically in loading datasets and handling variable types, were dominant for GPT-4o. In contrast, newer models like GPT-5 and o3 demonstrate higher stability with fewer environment and setup crashes,

reducing the total volume by nearly 80% and effectively mitigating environment and data handling issues.

8 Discussion and Future Work

Our results highlight the strengths and weaknesses of LLM agents in replicating research claims in SBS domains. Across the models and execution settings, agents demonstrate strong capabilities in structured code execution and debugging. However, they exhibit substantially weaker performance in earlier stages that require locating replication data on the Internet. Importantly, successful execution does not always translate into correct interpretation. Even when agents are able to reach beyond the generation stage and produce numerical results, interpretation errors, deviations from the pre-registered plan, and subtle implementation differences can lead to incorrect decisions. This gap suggests that benchmarks focusing solely on execution success may overestimate an agent's real-world utility, highlighting the importance of evaluating multiple stages of the replication process rather than a single outcome.

ReplicatorBench highlights promising directions for the development and evaluation of AI research assistants. Because current models struggle to locate appropriate data resources, we call for future work on agent development, specifically focusing on the planning stages, developing more specialized tools and effective search strategies to construct new replication samples. Additionally, as ReplicatorBench focuses on observational studies in the SBS domains where data is web-retrievable, future work should develop benchmarks for experimental replication. In these scenarios, agents must navigate controlled settings to collect or generate primary data. Together, these directions move the literature toward autonomous, end-to-end AI assistants for research replication.

9 Limitations and Ethical Considerations

Limitations. We acknowledge several limitations of ReplicatorBench. First, the benchmark is constructed from a sample of 39 replication studies. This scale is constrained by the scarcity of high-quality, expert-documented replication efforts that span multiple research stages. For the extraction stage, we collected three expert-annotated extractions about the focal claim of each study and documented necessary web resources. For the generation and interpretation stages, a group of researchers independently conducted a replication for each study. These efforts followed the standard scientific process: preregistering a research plan, conducting the study, and submitting final reports alongside a replication package of code and data. ReplicatorBench, thus, represents a deliberate design choice to prioritize data quality and process fidelity. We also recognize the shortcomings of using LLM-as-a-judge (LLMEval) for grading open-ended text, treating the rubric score as approximations rather than an absolute measure of replication competence. To mitigate the uncertainty of LLMEval, we used two complementary evaluation strategies. First, in extraction, we use LLMEval to assess our annotators' extractions in a leave-one-out setting to establish an upper bound. Second, we provide binary classification metrics for replication outcomes, which exhibit assessment patterns similar to LLMEval in the interpretation stage.

Ethical Considerations. We address several ethical considerations regarding data privacy, consent, bias, and potential misuse. Regarding data privacy and consent, all studies and replication packages in ReplicatorBench were sourced from the SCORE program with appropriate licenses (see §4). We ensured data diversity in our benchmark by selecting 19 papers from six disciplines in the social sciences (see Table 2). Regarding potential misuse, we recognize the risks of allowing LLM agents to autonomously write and execute code. We addressed this concern by setting up a sandboxed environment for each run and each study to isolate agent actions from host systems (see §6). We also adopted a human-check procedure for all executable commands and file-writing actions and observed no significant unsafe actions from the agents.

10 Conclusion

We introduced ReplicatorBench, a benchmark for evaluating LLM agents' capabilities to replicate research claims in SBS domains. Unlike prior benchmarks that focus mainly on execution under ideal conditions, ReplicatorBench breaks down the workflow into multiple stages, mimicking the human replication workflow. We grounded each stage in expert-led human replication efforts with a preregistered plan and final reports, which allows us to evaluate an agent's behavior at each stage rather than relying on only the final outcome. We developed a baseline agentic framework, ReplicatorAgent, and evaluated it against ReplicatorBench under multiple underlying LLMs and configurations. Our results show that although state-of-the-art LLMs are often capable of performing complex computational workflows and iteratively resolving execution failures, this performance does not consistently yield correct replication judgments. Furthermore, agents struggle in earlier stages, particularly in locating replication data on the Web, implying the importance of moving towards benchmarks that capture the entire replication workflow rather than just the outcomes.

Acknowledgments

This project is funded by Coefficient Giving as part of its 'Benchmarking LLM Agents on Consequential Real-World Tasks' program. We thank Anna Szabelska, Adam Gill, and Ahana Biswas for their annotation of the ground-truth post-registrations for the extraction stage.

References

[1] Nazanin Alipourfard, Beatrix Arendt, Daniel M Benjamin, Noam Benkler, Michael Bishop, Mark Burstein, Martin Bush, James Caverlee, Yiling Chen, Chae Clark, et al. 2021. Systematizing confidence in open research and evidence (SCORE).

[2] Abel Brodeur, Derek Mikola, Nikolai Cook, Thomas Brailey, Ryan Briggs, Alexandra de Gendre, Yannick Dupraz, Lenka Fiala, Jacopo Gabani, Romain Gauriot, et al. 2024. *Mass Reproducibility and Replicability: A New Hope.* Technical Report. The Institute for Replication (I4R).

[3] Colin F Camerer, Anna Dreber, Eskil Forsell, Teck-Hua Ho, Jürgen Huber, Magnus Johannesson, Michael Kirchler, Johan Almenberg, Adam Altmejd, Taizan Chan, et al. 2016. Evaluating replicability of laboratory experiments in economics. *Science* 351, 6280 (2016), 1433–1436.

[4] Open Science Collaboration. 2015. Estimating the reproducibility of psychological science. *Science* 349, 6251 (2015), aac4716. arXiv:https://www.science.org/doi/pdf/10.1126/science.aac4716 doi:10.1126/science.aac4716

[5] Chuxuan Hu, Liyun Zhang, Yeji Lim, Aum Wadhwani, Austin Peters, and Daniel Kang. 2025. REPRO-BENCH: Can Agentic AI Systems Assess the Reproducibility of Social Science Research?. In *Findings of the Association for Computational*

Linguistics: ACL 2025. Association for Computational Linguistics, Vienna, Austria, 23616–23626.
[6] Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. 2025. ResearchCodeBench: Benchmarking LLMs on Implementing Novel Machine Learning Research Code. *arXiv preprint arXiv:2506.02314* (2025).
[7] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out.* Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013/
[8] Ziming Luo, Zonglin Yang, Zexin Xu, Wei Yang, and Xinya Du. 2025. Llm4sr: A survey on large language models for scientific research. *arXiv preprint arXiv:2501.04306* (2025).
[9] Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. 2024. Discoverybench: Towards data-driven discovery with large language models. *arXiv preprint arXiv:2407.01725* (2024).
[10] B. Nguyen and D. Soós. 2026. ReplicatorBench: Benchmarking LLM Agents for Replicability in Social and Behavioral Sciences: Supplementary Artifact. doi:10.5281/zenodo.20506946
[11] Brian A Nosek and Timothy M Errington. 2020. What is replication? *PLoS biology* 18, 3 (2020), e3000691.
[12] National Academies of Sciences, Medicine, Policy, Global Affairs, Board on Research Data, Information, Division on Engineering, Physical Sciences, Committee on Applied, Theoretical Statistics, et al. 2019. *Reproducibility and replicability in science.* National Academies Press.
[13] Shuo Ren, Pu Jian, Zhenjiang Ren, Chunlin Leng, Can Xie, and Jiajun Zhang. 2025. Towards scientific intelligence: A survey of llm-based scientific agents. *arXiv preprint arXiv:2503.24047* (2025).
[14] Zachary S Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebl, and Arvind Narayanan. 2024. Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363* (2024).
[15] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. 2025. PaperBench: Evaluating AI's Ability to Replicate AI Research. *arXiv preprint arXiv:2504.01848* (2025).
[16] Minyang Tian, Luyu Gao, Shizhuo Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, et al. 2024. Scicode: A research coding benchmark curated by scientists. *Advances in Neural Information Processing Systems* 37 (2024), 30624–30650.
[17] Lukas Twist, Jie M Zhang, Mark Harman, Don Syme, Joost Noppen, and Detlef Nauck. 2025. LLMs Love Python: A Study of LLMs' Bias for Programming Languages and Libraries. *arXiv preprint arXiv:2503.17181* (2025).
[18] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.
[19] Emily C Willroth and Olivia E Atherton. 2024. Best laid plans: A guide to reporting preregistration deviations. *Advances in Methods and Practices in Psychological Science* 7, 1 (2024), 2515245923121380 2.
[20] Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. 2025. Scireplicate-bench: Benchmarking llms in agent-driven algorithmic reproduction from research papers. *arXiv preprint arXiv:2504.00255* (2025).
[21] Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, George Michalopoulos, Yue Zhang, et al. 2025. LMR-BENCH: Evaluating LLM Agent's Ability on Reproducing Language Modeling Research. *arXiv preprint arXiv:2506.17335* (2025).
[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. [n. d.]. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.
[23] Christine Ye, Sihan Yuan, Suchetha Cooray, Steven Dillmann, Ian LV Roque, Dalya Baron, Philipp Frank, Sergio Martin-Alvarez, Nolan Koblischke, Frank J Qu, et al. 2025. ReplicationBench: Can AI Agents Replicate Astrophysics Research Papers? *arXiv preprint arXiv:2510.24591* (2025).
[24] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675* (2019).
[25] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.

A Error Analysis

We categorize the execution failures into **Setup** (environment/dependency failures), **Input** (loading or formatting issues), **Implementation** (logic or syntax errors), and **Timeout** errors.

A.1 Quantitative Results

To classify these errors, we developed an automated parsing script that processes the raw execution logs for each run for each model. The script utilizes regular expressions to detect Python tracebacks, R execution halts, and standard error (*stderr*) streams within the agent's JSON observation fields Errors are then mapped to categories based on keyword signatures. While this heuristic approach allows for scalable analysis across the logs, it is an approximation. Ambiguous errors, like a file missing because the previous step failed vs. it actually not existing, are categorized based on the immediate error signal.

As shown in Table 5, **Input data errors** were the most prevalent category across all models, accounting for the vast majority of failures in GPT-4o runs ($N = 124$). Qualitative inspection suggests that these were primarily driven by the model's inability to robustly handle diverse file formats or strict datatype requirements in the provided datasets. **Implementation errors** were the secondary cause for GPT-4o ($N = 33$), but presented the primary bottleneck for the smaller GPT-5-mini ($N = 74$), which struggles more with correct syntax and library usage than data handling. Notably, GPT-5 and o3 achieved a massive reduction in total failures, with GPT-5 encountering only 6 Input data errors compared to GPT-4o's 211, highlighting a substantial improvement in the model's ability to navigate complex data environments without crashing.

Error Category	GPT-4o	GPT-5	GPT-5-mini	o3
Setup Errors	35	21	4	9
Input Data Errors	124	6	0	1
Implementation Errors	33	11	74	25
Timeout Errors	1	0	0	0
Total	193	38	78	35

Table 5: Distribution of runtime errors across models for 19 observational studies in ReplicatorBench. GPT-4o failures are dominated by Input Data Errors (data loading, type mismatches), whereas GPT-5-mini primarily exhibits Implementation Errors (coding logic and syntax).

A.2 Qualitative Analysis

A key design feature of our ReplicationAgent is *iterative debugging*, which is the ability to recover from execution failures. We present two contrasting cases to illustrate the capabilities and limitations of this approach.

Case 1: Successful Environment Recovery. Figure 5 demonstrates how GPT-4o was able to resolve a Docker failure caused by an incompatible library dependency. Rather than terminating, the agent utilized the error trace to diagnose the specific version conflict, proposed a targeted edit to the configuration file (*replication_info.json*), and successfully rebuilt the Docker image to achieve a successful run. This confirms the agent's ability to handle explicit errors.

Case 2: Silent Failures in Semantic Translation. While iterative debugging fixes crashed, it cannot always ensure correct replication results. Figure 6 shows a failed R-to-Python translation where the

```
---Turn 2 ---
Thought: The Dockerfile has been successfully generated.  The next step
      is to build the Docker image using this Dockerfile.
Action: orchestrator_build_image: "./data/original/4/python"
PAUSE
Running Action: orchestrator_build_image [Checkpoint: 2. Build Image]

--- Turn 3 ---
Observation: Docker build failed. See 'build_log' for details (pip
      dependency mismatch)
Build_log (excerpt):
ERROR: Ignored the following versions that require a different python
      version: ...
ERROR: Could not find a version that satisfies the requirement scipy
      ==1.7.1
... [built output truncated] ...
Thought: The Docker build failed because the specified version of `scipy
      ` (1.7.1) is not compatible with the Python version in the Docker
      image. I need to update the `replication_info.json` to specify a
      compatible version of `scipy` for Python 3.10. I will choose a
      version of `scipy` that is compatible with Python 3.10, such as
      `1.7.3`, which is available according to the error log.
Action: read_file: "./data/original/4/python/replication_info.json"

--- Turn 4 ---
Thought: I need to update the `replication_info.json` to specify a
      compatible version of `scipy` for Python 3.10. Based on the error
      log, `scipy==1.7.3` is a compatible version. I will edit the `
      replication_info.json` to update the `scipy` version and then
      regenerate the Dockerfile.
Action: edit_file: { "file_path": "./data/original/4/python/
      replication_info.json", "edit_type": "replace","old_text": "\"
      scipy==1.7.1\"", "new_text": "\"scipy==1.7.3\""}
[AGENT PROPOSES A FILE EDIT]
FULL PATH: /data/original/4/python/replication_info.json
DIFF:
--- /data/original/4/python/replication_info.json (before)
+++ /data/original/4/python/replication_info.json (after)
@@ -42,7 +42,7 @@
        "packages": {
          "python": [
            "pandas==1.3.3",
-          "scipy==1.7.1",
+          "scipy==1.7.3",
            "numpy==1.21.2"
Do you approve this edit? (yes/no): yes
Successfully edited /data/original/4/python/replication_info.json
...
Successful Run
```

Figure 5: Execution trace for resolving a Docker build failure caused by an incompatible SciPy version.

agent initially crashed due to a missing variable (*'data_clean'*). It correctly "fixed" the crash by adding code to load the dataset, but it hallucinated the filename (*'data_clean.rds'* instead of *'data_clean_5pct.rds'*). Because the incorrect file existed in the directory, the code ran successfully, leading to a "valid" execution log but incorrect replication conclusion.

B Evaluation Details

B.1 Effectiveness of LLM-as-judge

We rely on four human evaluators to score a subset of 53 extractions from a set of six papers. Similarly to the LLM judge, the human evaluators are given the agent-produced post-registration and expert-annotated post-registration to assign an interger score of 0 to 3 for each extracted field based on semantic match with the reference. We also provide both the human evaluators and the LLM judge with

```
============================ DESIGN STEP ============================
---Turn 7 ---
Thought: [...] The dataset seems suitable for replication as it includes
       the necessary variables.
Next, I will examine the R script `data_analysis_code.R` [...]

Running Action: read_file: {"file_path": "[...]/data_analysis_code.R"}

--- Turn 8 ---
Observation: [...]
data_clean_5pct <- readRDS("data_clean_5pct.rds")
# imputed data
data_imp_5pct <- readRDS("data_imp_5pct.rds")

m1 <- lmer(trstprl_rev ~ imm_concern + happy_rev + stflife_rev +
      sclmeet_rev + distrust_soc +
              stfeco_rev + hincfel + stfhlth_rev + stfedu_rev +
              [...])
[...]

Thought: [...]To proceed with the replication study, I need to translate
       this R script into Python, ensuring compatibility with the
      dataset and maintaining the original analysis logic [...]

I will start by writing a new Python script file and then proceed to
      translate the R code into Python.
Action: write_file: {"file_path": [...]/data_analysis_code__py.py", "
      file_content":

import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

AGENT FAIL TO LOAD DATASET DURING DESIGN STEP
# Load the dataset
# Assuming the dataset is already loaded in a DataFrame named '
      data_clean'

# Main analysis (complete cases, weights)
model = smf.mixedlm('trstprl_rev ~ imm_concern + happy_rev + stflife_rev
       + sclmeet_rev + distrust_soc +
                     stfeco_rev + hincfel + stfhlth_rev + stfedu_rev +
                    [...]])
result = model.fit()
print(result.summary())
}
[...]

========================== EXECUTION STEP ==========================
---Turn 24 ---
Observation: line 12, in <module>\n    data=data_clean, groups=
      data_clean['cntry'], weights=data_clean['pspwght'])
Name
Error: name 'data_clean' is not defined\n

Thought: [...] I need to load the dataset and assign it to `data_clean`
      at the beginning of the script. [...]

Action: edit_file: {
  "file_path": [...]/data_analysis_code__py.py",
  "edit_type": "insert_after",
  "anchor": "import statsmodels.formula.api as smf\n",
  "insert_text":
  # Load the dataset
result = pyreadr.read_r('/workspace/replication_data/data_clean.rds')
      Load the wrong data file.
}
[...]
Successful Run But Fail to yield the correct replication results
```

Figure 6: Design and execution trace of an incorrect translation attempt from R to Python.

multiple examples of each score 0, 1, 2, and 3. We ensure that each extracted field has three annotations. We measure the correlation between the human evaluators and the LLM-judge, detailed in Table. 6. We also consider more cost-effective metrics like ROUGE-L [7] and BERTScore [24]. According to table 6, LLMEval has the highest alignment with human judgement of semantic matching between the agent-produced post-registration and expert-annotated post-registration. Thus, we adopted LLMEval with GPT-4o as the main automated evaluation mechanism.

Method	Spearman's correlation.	Kendall's Tau
ROUGE-L	74.66	60.57
BERTScore	72.38	57.97
LLMEval (GPT-4o)	**86.18**	**78.43**

Table 6: Correlation between human assessments and automated evaluation metrics for extration stage, as indicated by Spearman ρ and Kendall τ correlation coefficients. The highest scores are highlighted with bold markers.

For design, execution, and interpretation, we collect human evaluations of 128 checkpoints across the three stages. Each checkpoint is annotated by three human evaluators with research experience, whose annotations achieve a Krippendorff's Alpha of 0.591, indicating moderate agreement and reflecting the complexity of evaluating scientific replications. We compare these human judgments with GPT-4o, and GPT-5 judgments to test whether the choice of LLM affects the resulting scores and whether GPT-4o received an inflated score when judged by itself. GPT-4o aligns closer with human judgment than GPT-5, with a Pearson correlation coefficient of 0.659 compared to 0.525 for GPT-5. The correlation between GPT-4o and GPT-5 judgments is lower at 0.403, suggesting that the choice of LLM can meaningfully affect automated evaluation. We therefore use GPT-4o as the main LLM judge and include GPT-5 scores only as robustness check.

B.2 Evaluation prompts and templates

For evaluation of the extraction, generation-design, and interpretation stages, we perform a single call to the LLM. For the generation-execution stage, the evaluation criteria require an inspection of the execution logs, so we adopt a ReAct [22] implementation for the GPT-4o judge. The full evaluation prompts, checkpoints, and templates are provided in our supplementary artifact [10].

B.3 Additional Results

Model	Extract - Info	Design	Execute	Interpret
o3	**60.25**	**54.16**	88.35	72.11
GPT–4o	55.93	48.79	71.47	53.40
GPT–5–mini	59.65	42.65	64.43	73.58
GPT–5	58.91	53.95	**94.26**	**77.42**

Table 7: Performance of our LLM agents across different stages against ReplicatorBench in Python mode. GPT-5 is used as the backbone model for LLMEval. All evaluation scores are normalized to a 0-100 scale. Macro aggregations are used. Results for *Extract - Web* and *Outcome* are deterministic and therefore remain the same as those reported in Table 3.

Model	Code Mode	Design	Execute	Interpret	Outcome
o3	native	78.61	69.01	87.82	55.56
	python	**84.57**	**95.10**	**95.48**	**62.35**
GPT–4o	native	78.47	60.33	76.31	45.45
	python	**81.14**	**74.10**	**77.01**	**59.82**
GPT–5	native	81.36	77.96	**91.18**	74.54
	python	**83.10**	**94.44**	77.73	**84.17**

Table 8: Performance of ReplicatorAgent in a Python-only setting compared to Native setting. LLMEval scores are reported for design, execution, and interpretation stage. Macro F1 Scores are reported for the final replication outcome (criteria met or unmet). The best results for each model are highlighted in bold markers.

C Task Templates, Agent Prompts, and Artifacts

To support reproducibility while respecting the page limit, we provide all prompts and structured output schemas in a supplementary artifact.[1] It contains the core ReAct agent system prompt, each specific prompts for each stage and structured output schemas for each stage in our pipeline.

We provide an example for the *alias matching* technique in the Extraction stage. If the ground truth URL is

https://www2.census.gov/programs-surveys/popest/datasets/2010-2019/counties/totals/,

then we accept

https://www2.census.gov/.

[1] https://doi.org/10.5281/zenodo.20506946